'68' MICRO JOURNAL

$2.95USA

Australia	A $ 4.75	New Zealand	NZ $ 4.75
Singapore	S $ 9.45	Hong Kong	H $23.50
Malaysia	M $ 9.45	Sweden	30:-SEK

VOLUME V ISSUE V • Devoted to the 68XX User • May 1983

"Small Computers Doing Big Things"

YOUR CHOICE-smart either way

- Over 140 software driven functions
- 82 x 24 or 82 x 20 screen format — software selectable
- High resolution 7 x 12 matrix characters — P-31 green phosphor
- Upper/lower case character set — plus graphics character set
- 56-key alphanumeric keyboard — plus 12-key cursor, numeric pad
- Internal editing functions — insert, delete, scroll, roll, slide, etc.
- Parallel printer I/O port
- 50 to 38,400 baud operation — programmable
- Cursor type, cursor position, print control characters, protected fields, shift inversion, dual intensity and many other features

8212 — twelve-inch diagnonal screen or **8209** — nine-inch diagnonal screen

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

The UniFLEX™ Operating System extracts every last drop

from the 8 bit 6809 microprocessor allowing it to outperform many 16 bit systems

With the UniFLEX™ Operating System, the 8 bit 6809 microprocessor can perform as well as larger CPUs in a multi-user, multi-tasking environment.

Independently developed from the ground up, UniFLEX™ closely models the features found in the UNIX™ Operating System. And in two years of use, UniFLEX™ has proven the abilities of the 6809 to perform large system functions when incorporated into a properly designed mainframe.

Some of the features supported include:

- full multi-user, multi-tasking capabilities
- hierarchical file systems
- device independent I/O
- four Gigabyte disk capacities
- full file protection
- inter-task communication via pipes
- I/O redirection
- task swapping for efficient memory usage
- full random-access files
- comprehensive shell command language
- foreground-background jobs
- electronic mail and printer spooling
- system accounting facilities

The support software currently available for use under UniFLEX™ is extensive. A sampling of the programs available includes:

- native C compiler (full implementation)
- native Pascal compiler
- FORTRAN 77 ANSI Subset compiler
- COBOL compiler with ISAM files, Report Writer & Sort/Merge
- Extended BASIC interpreter
- Extended BASIC precompiler
- text editing and processing software
- enhanced printer spooler
- variety of absolute and relocatable assemblers
- debug and diagnostic packages

Technical Systems Consultants, Inc. also offers a line of single user FLEX™ software products for 6800 and 6809 processors. For those having an absolute need for a 16 bit processor, UniFLEX™ will be available through OEM licensing arrangements for the 68000 microprocessor. Please call or write for additional information on individual products or OEM licensing arrangements.

UNIX™ is a trademark of Bell Laboratories.
FLEX™ and UniFLEX™ are trademarks of Technical Systems Consultants, Inc.

technical systems consultants, inc.

111 Providence Road
Chapel Hill, North Carolina 27514
(919) 493-1451

'68' MICRO JOURNAL

Portions of the text for 68 MICRO JOURNAL was prepared using the following furnished hard/software.

COMPUTERS-HARDWARE
Southwest Technical Products
219 W. Rhapsody
San Antonio, Texas 78216
SOS-5/8 DMF disk-COSI-8212W-Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe-OS9-FLEX-Assorted Hardware

EDITORS-WORD PROCESSORS
Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX-Editor-Processor

Great Plains Computer Company, Inc.
PO Box 916
Idaho Falls, ID 83401
STYLO-Mail Merge

CONTENTS

Send All Correspondence To:

Computer Publishing Center
68 MICRO JOURNAL
5900 Cassandra Smith
PO Box 849
Hixson, TN 37343
615 842-4600

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, Tenn. and additional entries. Postmaster: send Form 3579 to 68' Micro Journal, PO Box 849, Hixson, Tennessee.

SUBSCRIPTION RATES
USA
1-Year $24.50 2-Years $42.50 3-Years $64.50
FOREIGN
See Page 52

Items Submitted for Publication

Articles submitted for publication should be accompanied by the authors full name, address, date and telephone number. It is preferred that articles be submitted on either 5 or 8 inch diskette in TSC Editor format or STYLO format. All diskettes will be returned.

The following TSC Text Processor commands ONLY should be used (due to our proportional processor): .sp space, .pp paragraph, .fl fill and .nf no fill. Also please do not format within the text with multiple spaces. The rest we will enter at time of editing.

STYLO commands are all acceptable except the .pg page command, we print edited text files in continous text.

All articles submitted on diskettes should be in TSC FLEX™ format, either FLEX2 6800, or FLEX9 6809 any version.

If articles are submitted on paper they should be on white 8X11 bond or better grade paper. No hand written articles (hand written or drawn art accepted). All paper submitted articles will be photo reproduced. This requires that they be typed or produced with a dark ribbon (no blue), single spaced and type font no smaller than 'elite' or 12 pitch. Typed text should be approximately 7 inches wide (will be reduced to column width of 3 1/2 inches). **Please use a dark ribbon!**

All letters to the editor should also comply with the above and bear a signature. Letters of 'gripes' as well as 'praise' are solicited. We attempt to publish all letters to the editor verbatim, however, we reserve the right to reject any submission for lack of 'good taste'. We reserve the right to define what constitutes 'good taste'.

Advertising: Commercial advertisers please contact the 68 Micro Journal advertising department for current rate sheet and requirements.

Classified: All classified must be non-commercial. Maximum 20 words per classified ad. Those consisting of more than 20 words should be figured at .35 cents per word, 20 words or less $7.50 minimum, one time, paid in advance. No classified ads accepted by telephone.

Intelligent Serial I/O Processor Board Now Available

The GIMIX Intelligent Three-port RS-232C Serial I/O Interface can significantly increase throughput of a multi-user system by reducing the number of interrupts between user terminals and the host CPU. The Intelligent I/O Board accomplishes this by buffering data transfers between system and users and preprocessing of the data.

Appropriate on-board software and operating system drivers are required. Software and drivers for OS-9 Level 2 will be available shortly from GIMIX.

- INDEPENDENT ON-BOARD **2MHZ 68B09** CPU
- UP TO **20K** OF ON-BOARD MEMORY (EPROM and RAM)
- BUFFERED DATA TRANSFER BETWEEN HOST AND ON-BOARD CPU USING A **Z8038** FIO I/O INTERFACE UNIT
- THREE **RS-232C** SERIAL I/O PORTS **(6551As)** WITH SOFTWARE SELECTABLE BAUD RATES, WORD LENGTH, STOP BITS, PARITY

Standard Version Including 4K RAM (Without Software) $438.11

— ***PARALLEL VERSION COMING SOON*** —

Uniflex For GIMIX Winchester Systems

TSC will be providing UniFLEX compatible with GIMIX Winchester systems. The NEW versions of UniFLEX for use with the Winchester systems will be delivered on 5" media as well as 8" media.

GIMIX 30 Pin Prototyping Board Now Available

- Double sided with plated thru holes and gridded power and ground lines.
- 8 rows of pads on .100 x .300 centers: up to 41 fourteen pin ICs.
- Accepts standard 6, 8, 14, 16, 20, 24, and 40 pin DIP devices.
- The entire top edge has pads for .100 x .100 header (ribbon) connectors.
- Pads for solder connections or .100 center headers on all 30 bus lines.
- Accepts 3 TO-220 regulators, 1 on the +8V & 1 ea. on the +/– 16V lines.
- Provisions for decoupling caps distributed throughout the array.
- Can be used with wire wrap, wiring pencil, solder wiring, etc.

With gold bus connectors and heat sinks (unassembled) $38.33

Now Available From GIMIX

(U.S. & Canada Only)

THE WINDRUSH EPROM PROGRAMMER

- ★ Probably the most versatile EPROM PROGRAMMER available. Interface & software for EXORcisor - II (fully addressable) and S-50 bus systems.
- ★ PROGRAMS AND VERIFIES 2508/2708, 2516/2716 (SINGLE AND TRI-VOLT TYPES) 2532, 2732, 2732A, 2564, 2764 and the 128K TMS2528 (16K x 8) - - - - - - - WITHOUT ADDITIONAL 'PERSONALITY' MODULES - - - - - - -
- ★ PROGRAMMER extends out to your work area via 5' of twisted pair cable.
- ★ EXTENSIVE COMMANDS MENU . . . MOVE DATE, READ, PROGRAM, VERIFY EPROMS, EXAMINE/CHANGE BUFFER, FORMATTED DUMP OF BUFFER, FILL BUFFER.
- ★ Fully documented user's manual w/schematics & theory of operation. Professionally finished PCBs w/solder resist & component overlay.
- ★ SOFTWARE AVAILABLE FOR FLEX 2/9, SSB, OS-9 (LVL 1 NOW, LVL 2 LATER) and MDOS . . . All source files supplied. Specify disk size please!

NOTE: One version is supplied FREE. Extra versions: $25.00 each.

S-30 Interface/Programmer/Baseplate/Cable $375.00
EXORcisor Interface/Programmer/Baseplate/Cable $395.00

1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510
TWX 910-221-4055

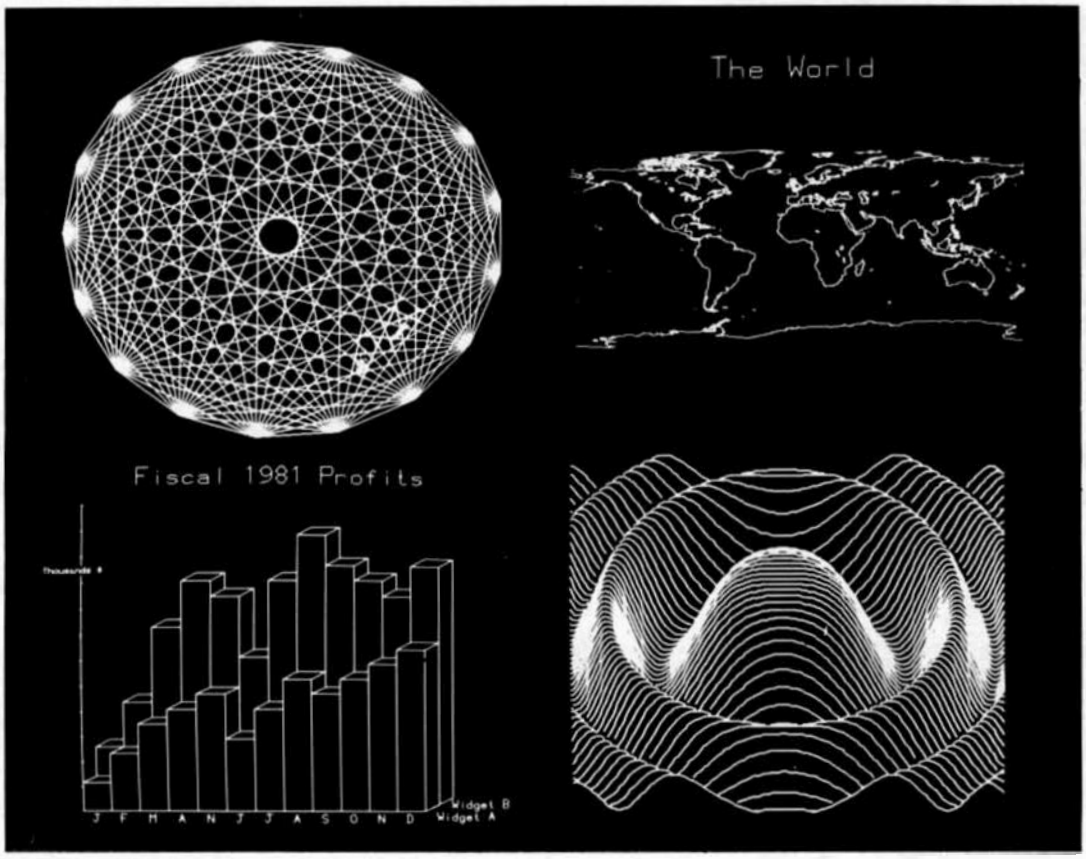
GIMIX INC.

ANNOUNCING ElectroScreen™

only $595

the Superior Alternative

to the Traditional Alphanumeric Terminals



The ElectroScreen™ Intelligent Graphics Board Features:

Graphics

- 512 x 480 resolution bit-mapped display
- Interleaved memory access — fast, snow-free updates

Intelligence

- 6809 on-board mpu
- 6K on-board firmware
- STD syntax high level graphics command set
- Removes host graphics software burden
- Flexible text and graphics integration
- Multiple character sizes
- User programs can be run on-board

Terminal

- Terminal emulation on power-up
- 83 characters by 48 lines display
- Easy switching among user-defined character sets
- Fast hardware scrolling

Additional Features

- SS-50C and SS-64 compatible board
- Board communicates with host through parallel latches
- Composite and TTL level video output
- 8 channel 8 bit A/D converter
- Board occupies 4 address bytes

See your dealer today!

The ElectroScreen manual is available for $10, credited toward purchase of the board.

The ElectroScreen has a 90 day warranty from purchase date.

Dealers, please contact us for our special introductory package.

Privac Inc (703) 671-3900
3711 S. George Mason Dr., Falls Church, Va. 22041

THE CHIEFTAIN™ 5¼-INCH WINCHESTER HARD DISK COMPUTER

SO ADVANCED IN SO MANY WAYS . . . AND SO COST-EFFECTIVE . . . IT OBSOLETES MOST OTHER SYSTEMS AVAILABLE TODAY AT ANY PRICE.

● HARD DISK SYSTEM CAPACITY

The Chieftain series includes 5¼- and 8-inch Winchesters that range from 4- to 60-megabyte capacity, and higher as technology advances. All hard disk Chieftains include 64-k memory with two serial ports and DOS69® disk operating system.

● LIGHTNING ACCESS TIME

Average access time for 5¼-inch Winchesters is 70-msec, comparable to far more costly hard disk systems. That means data transfer *ten-times faster* than floppy disk systems.

● 2-MHZ OPERATION

All Chieftains operate at 2-MHz, regardless of disk storage type or op rating system used. Compare this to other hard disk systems, no matter *how* much they cost!

● DMA DATA TRANSFER

DMA data transfer to-and-from tape and disk is provided for optimum speed. A special design technique eliminates the necessity of halting the processor to wait for data which normally transfers at a slower speed, determined by the rotational velocity of the disk.

● RUNS UNDER DOS OR OS-9

No matter which Chieftain you select . . . 5¼- or 8-inch floppy, or 5¼- or 8-inch Winchester with tape or floppy back-up . . . they *all* run under DOS or OS-9 with *no need* to modify hardware or software.

● UNBOUNDED FLEXIBILITY

You'll probably never use it, but any Chieftain hard disk system can drive up to 20 other Winchesters, and four tape drives, with a single DMA interface board!

● SMOKE SIGNAL'S HERITAGE OF EXCELLENCE

This new-generation computer is accompanied by the same *Endurance-Certified* quality Dealers and end-users all over the world have come to expect from Smoke Signal. And support, software selection and extremely competitive pricing are very much a part of that enviable reputation.

20-Megabyte Tape Streamer Back-Up Option

Available with all Chieftain hard disk configurations. This cartridge tape capability provides full 20-megabyte disk back-up in less than five minutes with just one command, or copy command for individual file transfers. Transfers data tape-to-disk or disk-to-tape. Floppy back-up is also available in a variety of configurations.

The Chieftain Computer Systems:

Here are the Chieftain 6809-based hard disk computers that are destined to change the data processing industry . . .

☐ **CHIEFTAIN 95W4**
4-megabyte, 5¼-inch Winchester with a 360-k floppy disk drive (pictured).

☐ **CHIEFTAIN 95XW4**
4-megabyte, 5¼-inch Winchester with a 750-k or-to-density floppy disk drive.

☐ **CHIEFTAIN 98W15**
15-megabyte, 5¼-inch Winchester with a 1-megabyte 8-inch floppy disk drive.

☐ **CHIEFTAIN 9W15T20**
15-megabyte, 5¼-inch Winchester with a 20-megabyte tape streamer.

Write or call today for details (including the remarkably low prices) on the total Chieftain Series . . . and on dealership opportunities.

SMOKE SIGNAL BROADCASTING®
31336 VIA COLINAS
WESTLAKE VILLAGE, CA 91362
TEL (213) 889-9340

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Telephone ( ) _____

ELEKTRA Means Quality

ELEKTRA CABINET Made of heavyweight 0.090" thick aluminum. Interior is 18-1/2" wide by 21-7/8" deep by 6-3/4" high. Heavy duty A.C. line cord. A.C. fuse holder. EMI filter. Fan with filter. Back panel has 10 cutouts for 'D' type data connectors. Front panel has key on/off power switch, 2 illuminated push button switches (Reset and NMI/Abort), and two cutouts for 5-1/4" disk drives. $250.00

Filler Plate for 5-1/4" drive opening: $10.00 Fan Filter: $10.00

POWER SUPPLY Highest quality linear power supply CONSERVATIVELY rated at 15a @ 8v, 3a @ 16v, 3a @ -16v. 3 primary inputs for light, rated, and heavy loading.
220v Version: $200.00 110v Version: $175.00

DISK REGULATOR BOARD WITH CABLES Standard version for 2 floppy drives: $50.00
Heavy duty version for 1 Winchester drive and 1 floppy drive. $75.00

ELEKTRA UNIVERSAL MOTHERBOARD Heavyweight 0.125" thick, 18" long by 9" wide, 11 memory (50 pin) slots. 4 or 8 slots may be cut off for shortening to 14" or 10" lengths respectively. 8 I/O (30 pin) slots. Complete address decoding of I/O slots. Choice of 4, 8, or 16 addresses per I/O slot. Base address for I/O slots can be placed at 32, 64, or 128 byte increments respectively. 1" spacing between all memory and I/O slots. Extended addressing capability for both memory and I/O ports for existing SS-50C bus specifications. On board baud rate generator with low and high ranges providing jumper selectable rates of 75 through 38,400 for each of the five baud rate lines. Slow device circuitry permitting 1MHz 30 pin disk controllers to run with 2 MHz 50 pin CPU boards.
Bareboard w/documentation: $80.00
Kit w/gold connectors: $320.00 Assembled w/gold connectors: $380.00
Kit w/tin connectors: $240.00 Assembled w/tin connectors: $300.00

ELEKTRA CHASSIS Includes cabinet, 110v power supply, power supply cables, standard disk regulator board with power cables, motherboard with gold square pin connectors assembled and tested. $850.00

ELECTRA CPU 6/9 Use either the 6802 or 6809 (to run 6800 software) or 6809. Has provision for up to 3 2716 EPROMs, 1K scratchpad, MC6840 triple timer, and an optional baud rate generator providing baud rates from 110 through 38,400 baud in two user selectable ranges. The board supports DMA by either HALT or BUSREQ when a 6809 CPU is used. The board will run any of the MIKBUG™ compatible monitors in the 6802/6808 mode and SBUG-E, HUMBUG, and MICROBUG in the 6809 mode. Versions of OS-9™ Level 1 are available.
Bareboard: $50.00 Kit: $225.00 Assembled: $275.00
2K 6800 HUMBUG: $40.00 2K 6809 MICROBUG: $30.00 See 4K HUMBUGS next page.
Optional Baud Rate Generator $25.00

ELEKTRA DPS DUAL PORT SERIAL CARD Fits the standard 30 pin SS-50 bus I/O slot. Can be configured for 4 addresses per port with the B port 2 addresses higher than the A port or for 16 addresses per port with the B port 4 addresses higher than the A port. Each port is terminated at two 16 pin dip sockets, one socket configured for modem and the other for terminal or printer. RTS, CTS, DTR, DCD and DTR are appropriately implemented. Each port has independent selection of baud rate. Each port allows the interrupt request to be independently jumpered to the IRQ or FIRQ/NMI bus line.
Bareboard: $20.00 Kit: $60.00 Assembled: $80.00
Cables (2 needed per board) Each: $20.00

ELEKTRA DPP DUAL PORT PARALLEL CARD Fits the standard 30 pin SS-50 bus I/O slot. Can be used in either the 4 or 16 addresses per I/O slot configuration occupying the first four addresses of the I/O slot. The direction of the TTL buffers can be controlled by either on board jumper connections or by a signal from the peripherals. The interrupt request line for each port may be individually jumpered to either the IRQ or FIRQ/NMI bus line.
Bareboard: $20.00 Kit: $60.00 Assembled: $80.00
Cables (two needed per board) Each: $20.00

ELEKTRA D-5 Dual drive cabinet for 5-1/4" drives with power supply, line cord, fuse, power switch, and power cable to drives 125.00
ELEKTRA HD-5 (Heavy duty version of D-5 package above) 150.00
ELEKTRA SHD-5 (Super heavy duty. Powers 1 Winchester and 1 floppy) 175.00
5' ribbon cable for dual 5 1/4" disk drives 40.00
ELEKTRA D-8 Dual drive cabinet, power supply, pa cable for 8" drives 350.00
6' ribbon cable for dual 8" disk drives 45.00

GOLD 10 PIN CONNECTORS (Specify male w/square pins or female) 1.50
TIN 10 PIN CONNECTORS (Specify male w/square pins or female) 0.50

Prices and inventory are subject to change without advance notice.
This ad is our catalog.

WARNING AAA Chicago Computer Center does not provide repair or diagnostic service for customer assembled kits. AAA Chicago Computer Center does warranty and maintain service for our assembled boards. The customer should carefully take into consideration the small differential separating our kit and assembled prices when making his choice of purchase.

We have introduced our line of computer equipment with the purpose of offering the highest quality of components possible at affordable prices. These products are intended for OEM applications where it is the responsibility of the purchaser to integrate these components with suitable memory, disk controllers, drives, and software along with I/O terminals to form working computer systems.

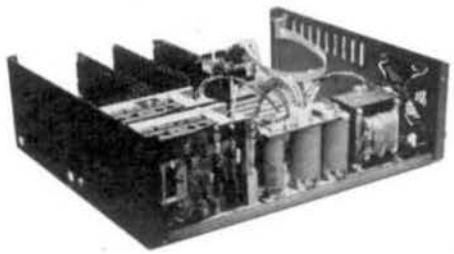
AAA CHICAGO COMPUTER CENTER
120 CHESTNUT LANE • WHEELING, IL 60090
(312) 459-0450
Technical consultation available 4 PM to 6 PM most weekdays. Closed evenings and weekends.

TERMS Minimum order $20.00 Shipping and handling estimates within the Continental U.S., add 3% (MINIMUM $2.50). Illinois residents add 6% sales tax. We will refund your overestimated shipping and handling charges. Foreign shipping and handling, add 10% (MINIMUM $10.00). Foreign orders must be prepaid in U.S. dollars. Heavy foreign items will be shipped air freight collect. Please phone between 4 PM and 6 PM weekdays if questions arise regarding shipping fees. Master Charge, Visa, and American Express honored.

Our apology. We are not staffed to answer technical inquiries through the mail. Please phone for technical help during the hours indicated above. The too frequent changing of our inventory and prices makes it uneconomical to publish a catalog. Our ads are intended to serve that purpose. Prices and inventory are subject to change without advance notice.

SD/DD DISK DRIVES	1 head	2 Heads	1 head	Ilippy	2 heads	2 heads
30 day guarantee	CDC	CDC	MPI	Siemens	MPI	QUME
5-1/4", 40 tracks	225.00	299.00	250.00	260.00	325.00	350.00
5-1/4", 80 tracks	N/A	375.00	325.00	335.00	425.00	450.00
Service Manual (Specify 40 or 80 track)					20.00	35.00
8", 77 tracks DS/DD Qume						525.00
Service Manual (Qume DT-8)						40.00

ELEKTRA
COMPUTER PRODUCTS



SMOOTH™ Software

(All of our software is copyrighted and all rights are reserved. Source is either supplied or optionally available at extra cost so that the purchaser can modify our programs for his own use. Licensing, however, is required for commercial resale.)

SUPER MODEM PROGRAM

Single character command for simplicity.

Hardware independent (No interrupts required). Assumes modem is connected to an MC6850 (serial interface) and the control terminal is connected to an MC6850 (or MC6820 when used with the video version of GMX UG™).

Transmit manually to distant computer.

Transmit disk files (text) of any length to distant computer.

Receive and save disk files (text) of any length on local disk system. If sending computer does not support an X-on/X-off protocol, then the received files are limited in size by the computer memory.

Tested for full duplex operation at speeds up to 9600 baud. (CRT terminal must be capable of operating at a baud rate higher than the one the modem is operated at.)

Half duplex option in case distant computer doesn't echo.

Echo option so user can simulate a time sharing system. (Super Modem Program doesn't support auto-answer but the source is provided for those individuals who wish to adapt our program to their special needs.)

Replaces CR with CR/LF (user option) for those using time sharing systems that don't transmit LF's.

Slow disk file transmit based on character verification (plus user installed timing loops if necessary) for use on time sharing systems to which disk files cannot be sent at speeds suggested by the baud rate.

Please specify 6800 or 6809, SSB or FLEX™, 5" or 8".
Manual and disk with both source and object code $75.00

STANDARD MODEM PROGRAM

Same as Super Modem Program above but without ECHO option, CR/LF for CR option, slow disk file transmit option, nor X-on/X-off option. Reception of disk files is limited to those small enough to completely fit within the receiving buffer.

Please specify 6800 or 6809, SSB or FLEX™, 5" or 8".
Manual with instructions, source listing, and flow chart; disk with both source and object code $45.00
Manual with instructions, source listing, and flow chart 25.00

MODEMS (By U.S. Robotics) Auto-Dial is Hayes compatible	MANUAL ANSWER	AUTO ANSWER	AUTO-ANS AUTO-DIAL
300/1200 Baud, direct connect	N/A	449.00	499.00
1200 Baud (120 cps), direct connect	349.00	399.00	N/A
300 Baud (30 cps), direct connect	149.00	179.00	N/A
300 Baud (30 cps), acoustic	119.00	N/A	N/A

THESE ADS ARE OUR CATALOG

GIMIX™ CLEARANCE SALE — ALL GIMIX™ ITEMS MUST GO!

All new GIMIX items 20% off list prices while supply lasts. (See our ad in the April issue of the MICRO JOURNAL for list pricing). Some items may be drop shipped by other former GIMIX dealers.

The following used items are in perfect working condition.
32K GIMIX memory boards $175.00. GIMIX 6800 CPU boards $125.00. GIMIX chassis from $450.00 to $500.00. GIMIX 64 X 16 video boards $100.00. GIMIX 80 X 24 video boards with character generators $250.00.

SMOOTH™ Software

ALL IN ONE
Editor - Text Processor - Mailing Labels
Mailing Lists - Use any CRT terminal and printer

Supports Editing commands such as bottom, change, delete, find, insert (single line), input (multiple lines), list, next, overlay (with cursor editing, character deletion and insertion), overstrike (for selected darker text), print, restart, set, top, underline, up, and verify.

Supports Text Processing commands such as block copy, block move, centering, margin justification (widen and narrow), paging, and tabbing.

Mailing Lists and Labels. Use the same mailing list disk file (with protected areas) for both mailing labels and repeat letters. Repeat letters are personally adressed to each person or selected persons on the mailing list.

Most Powerful File Handler found in any editor. Append one file to the end of another, or insert (merge) one file into another as designated by the line pointer. Print specified lines to your printer or to a disk file. Edit files larger than the text buffer. Does not produce output files when not desired. Delete disk files from the editor.

Printer commands. Control characters can be sent to the printer for form control either directly from the control terminal or by imbedding them in the text. The set command contains interface initialization and character output routines to support the SWTPC MP-C interface as well as the standard serial and parallel interfaces. Jumps are also provided to user supplied printer routines. User selects the port address (0 thru 7, A or B) thereby eliminating the need for the user to install printer software routines. Editor can be initialized for either 4 or 16 addresses per port.

Editor allows exiting to either the monitor or DOS and then reenter (Warm Start) without destroying previously prepared text in the buffer. The Restart command preserves contents in the buffer without the user having to reload the Editor.

The Editor allows the user to toggle between full duplex (no echo) and half duplex (echo) as needed. It responds to commands in both upper and lower case and can be used to create assembler source code and Basic programs as well as text.

Specify 6800 or 6809, SSB or FLEX™, 5" or 8"	75.00
Printed source listing is available for an additional	35.00
All-In-One Write'n Spell, and Spell'n Fix package	250.00

Software by Technical Systems Consultants, Inc.	UniFLEX™ w/1 yr. mnt.	FLEX™
DOS (includes Editor and Assembler)	550.00	150.00
Editor or Assembler		50.00
68000 Cross Assembler on 6809	300.00	250.00
6809 Cross Assembler on 6800 or 6800 FLEX™ Utilities		100.00
Text Processor or Sort/Merge Package or 6809 FLEX™ Utilities	150.00	75.00
Extended Basic	200.00	100.00
Basic Precompiler (specify standard or extended)	150.00	50.00
Pascal	300.00	200.00
Debug Package or Diagnostic Package	75.00	75.00
6809 Relocating Assembler & Linking Loader	575.00	150.00
Fortran (Requires Relocating Assembler & Linking Loader)	350.00	275.00
Fortran (With Relocating Assembler & Linking Loader)	450.00	375.00
Cobol	750.00	500.00

Software by Microware Systems Corp.	Run-Time Package	Update	Source	Manual Only	Object w/Man.
OS-9™ Level One Operating System		75.00	400.00	40.00	200.00
OS-9™ Level Two Operating System		75.00	N/A	40.00	500.00
BASIC09™	100.00	75.00	N/A	25.00	200.00
OS-9™ Macro Text Editor			300.00	15.00	125.00
OS-9™ Interactive Assembler			300.00	10.00	125.00
OS-9™ Interactive Debugger (Disk version)			100.00	10.00	50.00
CIS Cobol Compiler	600.00	50.00	N/A	80.00	900.00
Pascal Compiler	100.00	100.00	N/A	40.00	400.00
Microware yearly support service ($200.00 for OS-9 Level 2)					75.00

SPECIALS

* New Televideo Z80A, 4MHz, 64K computer system with 950 compatible built-in terminal, dual 5-1/4" floppy drives (1 Mbyte capacity total), CP/M	3295.00
* Same but with one floppy and one 10Mbyte Winchester	5495.00
* Six user versions available	C II
* Modems (Up to $100.00 off list). See our ad on the previous page.	
* SWTPC MP-09B, socketed, with baud rate generator (only 1 left)	175.00
* SWTPC MP-A2, socketed, all with crystal clock 1 MHz (only 4 left)	115.00
* SSB BFD Floppy DiskControllers (Version 3)	175.00
* MICROBUG (2K, 6809, Baby HUMBUG by Peter Stark)	30.00
* Cat 300 Baud Accoustic Modem (only 1 left)	99.00
* SWTPC 4K Memory $15.00, 8K Memory $40.00, MP-C $10.00	
* SWTPC Chassis,PS, MB combos (some with modified PS) $50.00 thru	125.00
* High speed tape reader $75.00 * SWTPC PR-40 Printer	75.00
* Percom 50 pin prototyping board $30.00 * 30 pin proto. board	$20.00
* Used MPI B51 and Siemens 5-1/4" floppy drives	125.00

Special Software

MICROBUG (2K, 6809 Baby HUMBUG by Peter Stark	30.00
4K 6809 HUMBUG	75.00
4K 6800 HUMBUG (RAM needed at $A000 and $D000)	65.00
2K 6800 HUMBUG (With cassette LOAD and PUNCH)	40.00
2K 6800 HUMBUG (Extra commands available on cassette software)	40.00
Other HUMBUG versions including video versions are available.	
Spell'n Fix by Peter Stark	178.58
Write'n Spell by Peter Stark	75.11
All-In-One, Spell'n Fix, and Write'n Spell package	250.00
Dynamite (Update. Send original diskette plus $40.00)	60.00
Dynamite + (Update. Send original diskette plus $40.00)	100.00
SUPER SLEUTH Disassembler System ($101.00 for OS-9 version)	99.00

SWTPC

6809 SWTPC FLEX™ Disk and manual (Disk only $15.00)			35.00
DC-4 Disk Controller (SS/DS, SD/DD, 5-1/4")			230.00
S32 Universal Static Memory Board (without memory chips)			124.50
SBUG-E (2716 compatible)	25.00	MP-R 2716 Eprom Programmer	114.50
MP-S2 Dual Port Serial	120.00	MP-L2 Duel Port Parallel	120.00
MP-N Calculator Board (kit)	54.95	MP-N (assembled)	92.00
MP-T Interrupt Timer	92.00	MP-09 2MHz 6809 CPU Board	295.00

Smoke Signal Broadcasting

DCB-4A Double Density Controller Board for 5" and 8" with DOS					549.00
SSB DOS (Specify 6800 or 6809, BFD or DCB-4A, 5" or 8")					75.00
SE92/SA92-5 (6809 Edit/Assm for DOS)					89.95
SSB Monitor (Specify 6800/6809, $8000/$E000/$F7E8)					75.00
SSB version of FLEX™ (without Editor and Assembler)					150.00
LMB-1A Motherboard					399.00
SCB-69 6809 CPU Board					399.00
PAR-1 Dual Port Parallel Board					89.00
SER-2 Duel Port Serial Board with 2 Cables					129.00
Chief 90 64K Computer System					2195.00
Chief 9524 64K Computer System with DS-DT-DD 5" FD					4325.00
Static Memory Boards	M-16-X	195.00	M-24-X	295.00	M-32-X 395.00
Dynamic Memory Boards	M-128-X	995.00	M-256-X	1295.00	M-512-X 1895.00

SMOOTH and **ELEKTRA** are trademarks of AAA Chicago Computer Center.
FLEX and **UniFLEX** are trademarks of Technical Systems Consultants, Inc.
OS-9 and **BASIC09** are trademarks of Microware Systems Corp.
GIMIX and **GHOST** are registered trademarks of GIMIX Inc.
CP/M is a registered trademark of Digital Research Inc.
Z80A is a registered trademark of Zilog, Inc.

THE BEST 30 PIN FLOPPY DISK CONTROLLER THAT YOU CAN BUY! — THE ELEKTRA SFC

Controls up to four 5 1/4" drives and four 8" drives for a total of eight system drives for both 6800 and 6809 systems. (FLEX software has a built in limit of a system mix of four drives). Improved programmed input/output (PIO) interface that allows single and double density 5 inch operation at 1MHz in both 6800 and 6809 systems. Allows single density 8 inch operation at 1MHz in both 6800 and 6809 systems. Allows both single and double density operation of both 5 inch and 8 inch drives at 2MHz in both 6800 and 6809 systems without slow device circuitry. Hardware and software compatible with existing SWTPC DC-1, DC-2, AND DC-3 type controllers. Controls virtually any floppy disk drive, any mix of drive size, recording density, track density, number of sides, motor control, with or without head load solenoids, including the new 2 Mbyte 5 inch drives. Analog phase locked loop data separators for maximum reliability. Provides separate adjustments for 5 inch and 8 inch drives. Analog write precompensation circuit with separate adjustments for 5 inch and 8 inch drives. Designed to meet the data hold requirements of Western Digital floppy disk controller IC. Schmidt trigger hysteresis input buffers for disk drive signals. Software write protect control. Interrupt/Data Request enable control. 12 option jumper areas to select just the right combination of hardware features to match your drives and software. Provides many separate option selections for 5-inch and 8 inch drive operation. Gold bus and drive cable connectors. Test connector for easy setup and field service. Single 5 volt power supply. 4 or 16 addresses per slot decoding. Ideal replacement for existing controllers to add double density data storage and 8 inch drive capability. Disk driver software and disk formatting utilities available to extend the capabilities of your current software. Get more for less with the ELEKTRA Super Floppy Controller.

ELEKTRA SFC Super Floppy Controller (assembled and tested)	$250.00
Disk with drivers and formatting utilities	$30.00
EPROM with driver overlays	$30.00
TSC General FLEX or FLEX 9 both with Editor and Assembler	$150.00

THE BEST WINCHESTER DISK SYSTEM THAT YOU CAN BUY! — THE ELEKTRA WINCHESTER

Has automatic error detection and **CORRECTION** of up to 11 bit burst errors.

SS-50 bus, extended addressing capabilities, DMA, on board sector buffer, operates with 6800 and 6809 type processors, drivers included for FLEX 9 or OS-9. Drivers for FLEX 2 (6800) available for an additional $100.00. Price includes host interface, controller, drive(s), and cables.

7.5 Megabyte ELEKTRA single drive Winchester system	$1995.00
15 Megabyte ELEKTRA dual drive Winchester system	$2995.00
19 Megabyte ELEKTRA single drive Winchester system	$2995.00
38 Megabyte ELEKTRA dual drive Winchester system	$4995.00
ELEKTRA heavy duty regulator bd. (powers 1 floppy and 1 Winchester)	$75.00
ELEKTRA SHD-5 cabinet, ps, ps cables (powers 1 floppy and 1 Winchester)	$175.00

AAA Chicago Computer Center . See our ad on the previous page to your left for ordering instructions.

Technical Consultation available 4 p.m. to 6 p.m. most weekdays. Closed evenings and weekends.

Flex User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

Clarifications

A few months ago I did a review of Windrush PL9 compiler, and made a remark (jokingly, I thought) about them using the algorithms I published in the November 1980 '68' Micro Journal, but not giving me credit for them. Windrush wrote me and asked me to look at that issue of '68' and tell them how they could have known that the algorithms were mine. I did dig out that issue and look, and I found out what the problem was. The algorithms were discussed in detail in my column in the previous (October 1980) issue, but there wasn't room for the listings. Don Williams added a little note that they would be published "next month". In the November issue, they were added to the end of my column, but there was no reference to them whatever anywhere in the text of that month's column, (not even a little "Here are the listings that wouldn't fit last month." note). I must therefore agree with the folks at Windrush that the authorship of the scientific functions was not clear at all. They have added a credit to their listings, and we are all happy. If my remark caused anyone to think that I was implying that Windrush had done anything wrong by using those functions, I'm sorry.

- -

Editor's Note: We here at 68 MICRO JOURNAL regret that this occurred (our ommission). WINDRUSH is a company that has dealt most fairly with our readers and we appreciate their fair and honest policies. We have never received a complaint concerning their products or customer satisfaction! Should any advertiser desire to 'quote' from 68 MICRO JOURNAL, a simple letter will normally get favorable results. So if you desire to reprint or use something from 68 MICRO JOURNAL, a simple letter will normally get approval. We do want to know; yet, we might also miss a 'credit', if so, we want to know that also.

- -

Along that line, any listings published in this column are understood to be public domain. I hadn't really thought much about anyone using them for commercial purposes before, but that doesn't change anything. They are still for anyone to use who wants to use them. I do appreciate users including a credit line for the source.

While on the subject of Windrush, let me say that I have had continuing correspondence with them regarding PL9. They have won my all time award for thorough answers to my questions, criticism, and suggestions for enhancements. Some folks have sent me software for testing and review, and then all but ignored any correspondence about that software from me. The folks at Windrush are definitely not in that category. In response to my questions, they have fixed two bugs and made several clarifications to their manual, which was pretty good in the first place as software manuals go.

Since writing the review update, I have had a chance to use PL9 in the latest version, rather extensively. I've translated some non-trivial programs from Pascal, which is a rather easy exercise, and found PL9 to be more code efficient than the best Pascal compiler that I have. Some of this is due to being able to edit some of the library functions and only include what is needed for a given program. What is more impressive is the single step and very rapid compile operation. When I am done with a PL9 program, I have the library files, of course, but only two files that are specific for a given program, the original source file and the binary file. Compilation was VERY fast, and if the compiler catches errors, it is easily put in the co-resident editor mode so a correction may be made quickly and the file re compiled without aborting the compiler-- loading the editor -- editing the file -- exiting the editor -- and re-loading the compiler.

Contribution Records

UDRI sent me their DBM for the Color Computer, and their Contributions Package (the latter at my request). Last year, I kept the records for my church's contributions, and prepared statements for tax purposes for all contributors. At the time, I wrote my own software in BASIC, and though it worked, there were several inconveniences in it that I would change if I used it again. Since I now have the UDRI package, I am going to use it this year, and I've promised UDRI a review after I've used it for a while so that I am familiar with it from first hand experience in actual use. I will be sending comments, suggestions, and reports on any bugs I find to UDRI, and will review the package in a few months, after I get well into using it.

One thing I like about it already, is that it uses Random files. Mine had used Sequential files, which presented a few problems, as in the case of the same contributor making multiple contributions on a given Sunday. Once a record was updated for a Sunday, it was written to the new sequential file, and it was not possible to update it again in a straightforward manner.

Spectral FLEX

If you are a regular reader of '68', you have most likely read Bob Nay's column and seen the short reviews of Steve Odneal's FLEX sold by Data Comp, and Frank Hogg's implementation of FLEX. The third supplier of FLEX for the CC, is Spectral Associates of Tacoma WA. They have a few wrinkles on FLEX that I should mention here. Spectral's version of FLEX may be run on a RS converted 32K color computer without opening the computer and voiding the warrantee (only if all the memory happens to consist good 64K chips) by means of a little adaptor board they call a supercharger. This board plugs into the interface slot and it contains a connector into which the disk interface is plugged. That is, this board is put between the interface and its connector in the CC. The supercharger brings about the access to the additional memory. Spectral supplies a test program called RAMTEST that will tell you if all the memory is good, and if it is, you have 32K of free memory.

Rapid developments in the CC have caused considerable complications, as evidenced from the following quote from the Spectral Manual.

"Version 1.26 of FLEX+ contains a version of FLEX+ which will function without a supercharger. This file is called FLEX.NOS and it must replace the current FLEX.SYS in order to become the resident operating system upon power up. To do this, you must first DELETE FLEX.SYS and finally LINK FLEX.SYS. (I think they mean FLEX.NOS here) You will then have a version of FLEX+ which does not require a supercharger. It will work on all TOP System 100 computers and TANDY revision F or NC Color Computers. CAUTION- this version of FLEX+ will not support Printer spooling."

"Version 2.0 of FLEX+ is a stock version which does not require a supercharger to function. It is limited as described above and contains a file called FLEX.SUP which may be converted into a supercharger dependent operating system (capable of print spooling) as described above."

Sounds a bit complicated. One point that may have you wondering, is the mention of printer spooling. Standard FLEX as it was supplied for the 6809 systems on the SS-50 bus, had a printer spooling feature. You would run the output of the text processor or the assembler to a disk file with the extension .OUT, and then you would invoke the PRINT command and tell the computer to print that file. The system would start the printer and give you the +++ FLEX prompt so you could continue using the computer with the printer running in the "background". You had to have a timer board installed in the computer to supply interrupt signals so that the printer could run as a background task.

Apparently, Spectral has figured out how to put the CC into this mode, and it works rather well, just like the original SS-50 version. Printer spooling is a convenience if you want to continue to use the computer for something else while printing a long listing. However, it does not operate without a penalty. While a disk is being accessed to get another sector's worth of information, the terminal essentially goes dead. If you are a touch typist and are typing along dumping text into the computer as I am doing now, you will probably find that the computer has ignored a couple of characters every time the disk was accessed. If you are just editing a source file for a program, and have your eye on the screen, you won't have much problem when running in this mode, but fast typists should be advised that there are some drawbacks. At any rate, Spectral FLEX is the only version offering printer spooling.

What's the Difference?

Perhaps it is about time here to compare the three

versions of FLEX that are available for the CC. First it should be said that they are about 95% identical. That is, they all work. They all have the standard FLEX utilities supplied by TSC. However, each has some feature or other that sets it apart from the others. Spectral, as mentioned above, has the printer spooling feature, and the possibility of use without modifying your older CC.

Data Comp has a couple of features not found in the other two. They have a utility called DISKEX that lets you examine the data in a disk sector, and even change it if you so choose. D.C. has a set of three utilities that allow you to move files back and forth between RS formatted disks and FLEX formatted disks. RSREAD will read a RS binary file to a FLEX disk (assuming you have two drives). RSCVBIN will convert that file to a standard FLEX binary file. It may be run in FLEX, modified, disassembled, etc. A FLEX binary file may be written to a RS disk with the RSWRITE utility. I would say that D.C. FLEX has the advantage for more experienced programmers who want to poke around in the RS operating system and perhaps write software to run on the CC in the RS operating system.

The FLEX from Frank Hogg Laboratory has some very nice software configuration capability. You can set up the system parameters most flexibly of the three systems using FHL's SETUP utility. For example, you can set up the printer output routine so that it supplies the linefeed after the carriage returns, which are required by a printer set up in the non-auto linefeed mode (standard in FLEX) or you can set up the printer routine to suppress linefeeds (standard RS system mode). If you want to run a printer ALWAYS on your CC, you would probably set it to the Auto Linefeed mode so it would work with the RS operating system, and configure the print routine in FLEX to the "R" mode (Radio Shack). You could then move back and forth between the systems without having to reconfigure the printer, which in the case of the EPSON involves removing four screws and opening the case to get at the DIP switch to change the LF mode.

FHL FLEX also allows configuration of the disk drives with regard to number of tracks, sides, and stepping rate and configuring the terminal with regard to several different high resolution formats, all of which include choice of "white on black" or "black on white" characters.

Incidentally, all three systems support a 51 character by 24 line screen format with black letters on a light background (normally beige). Due to limitations of color CRT's, all work best in that mode with the color turned down so that the display is essentially black and white. All three also supply an editor for use with FLEX. FHL's offering includes a more capable editor than the standard TSC one that comes with the other two systems. All three supply an assembler.

So, there you have it. One or the other of these systems might just have that one feature that you really "need". They all work very well.

Epson Converter

A few months ago I mentioned the possibility of a very low cost serial to parallel converter to run an Epson on the CC. Clay Abrams sent me a design, the schematic for which is reproduced here. I've built a couple in what I might call a "poor man's version". The Motorola 14411 is rather expensive, as is the crystal. I have substituted a 555 timer (35 cents). The 555 is a very stable timer IC. It is only as stable, however, as the parts used in the timing circuit. Build this using a ceramic capacitor and a carbon composition resistor, and I guarantee that you will always be "tweaking" the frequency. You must use a "film" capacitor at least a mylar type such as the Cornell Dubilier type WMF, and preferable one of the Mylar Polystyrene low temperature coefficient types. Then you must use a wirewound or metal (tin oxide) film resistor and a wirewound trim potentiometer. The values shown are for a 16X clock for 600 baud (9600 HZ). The CC outputs to its printer port at 600 baud unless you POKE the counter location with other values.

We built one of these, set its period with an oscilloscope, connected it, and it ran. The AY5-1013 is available from most Radio Shack stores in a 5 volt single supply version for around $5.00. The most expensive part is the connector for the EPSON at around $7. The Epson manual indicates that pin 35 is "pulled up" to +5 volts. It was an easy matter to modify the Epson board to connect the 5 volt supply directly to that pin and power the interface from the Epson with no problems encountered. The 5 volt only version of the AY5-1013 from RS, of course eliminates the requirement for -12 volts. Should you want to run the interface at a higher baud rate, use a correspondingly smaller timing capacitor. For example, for 1200 baud, use a .005 Mfd capacitor, etc.

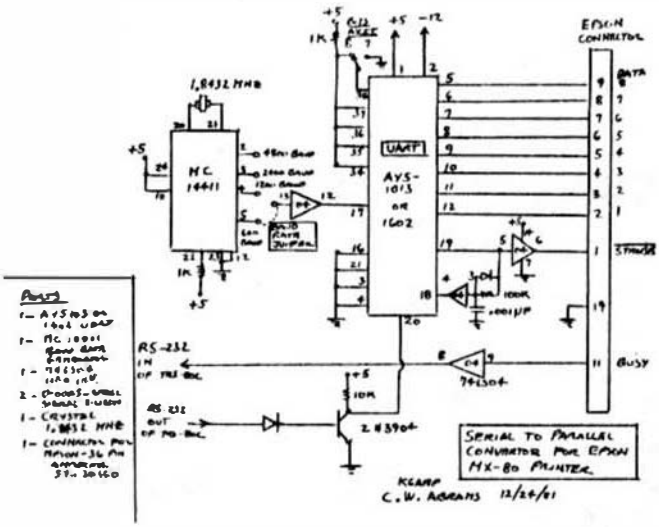
Should you be a "disbeliever" in RC timing circuits, go ahead and use the crystal and the 14411.

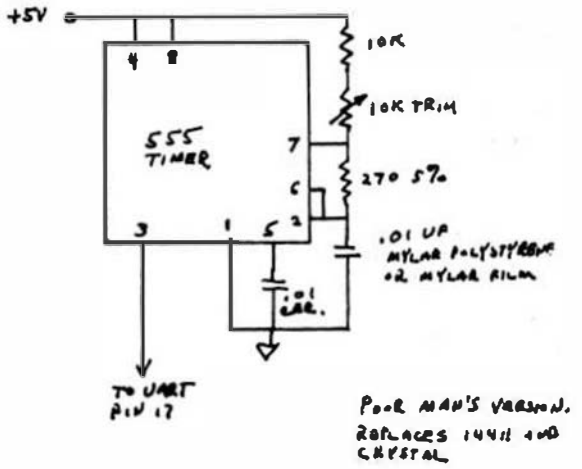
More on Disk Drives

A few months ago I made some comments on the use of a couple of 80 track double sided drives with the CC, as though there were no problems whatever with simply purchasing two such drives and using them. There is a SLIGHT problem with that. All of the software for the CC is supplied on 40 track (35 track for RS disks) format. I was speaking as a person with several 6809 FLEX computers around. I have available two 40 track double sided drives, two 35 track single sided drives, and a couple of 8" double sided ones, and I can therefore usually copy a disk in any format to any other. Worst case, I can juggle drives, and put one of the 35 or 40 track ones on the CC to boot up FLEX. It is an easy matter from that point, to copy FLEX to a new system disk on the 80 track drive.

The RS system always uses 35 tracks, single sided, and double density. Since the system doesn't know that the drive is double sided and 80 track, it will format the first 35 tracks on the first side, of the disk in the 80 track double sided drive. Not very efficient, but completely RS compatible. Again, the trick is to be able to read the disk supplied by the software supplier. (Games can be bought on cassette, and many may be loaded from cassette and saved to disk). WARNING. Not all of the games on the market may be copied from cassette to disk by this means. Some may, if you know or can figure out the load and transfer addresses. I was pleasantly surprised to find out that Spectral Associates games come on a cassette with complete instructions for copying them to a disk file!

All three of the FLEX suppliers have missed the boat on one small point. None allow you to set the disk drives to the double stepping mode, in which it would be possible to read a 40 track disk. I can conceive of a hardware solution to the problem, a doubler that would generate two step pulses to the disk drive for each one received from the disk controller. Such a circuit would make it possible to read any 40 track disk on an 80 track drive, and even boot FLEX from the supplied disk!





COLOR User Notes

Robert L. Nay
5900 Cassandra Smith Rd.
Hixson, Tn. 37343

Part 2: CROSS REFERENCE for the Color Computer EXTENDED BASIC ROM

Last month we printed a Cross Reference Listing for the Level 1, Ver. 1.1 BASIC ROM; this we are providing the same thing for the **EXTENDED BASIC ROM**. Again, it is fairly complete. Notable exceptions are that there are several locations within the EXTENDED BASIC ROM where an Indexed JSR off of the U, Y, or S register is made (at $9465, $94E2, $94FE, $99EC, $9A05, and $94EF to mention some of them). Also, $9A12 JSR's indirect from $00D9. There are a few locations where the Disassembler interprets a Compare #$$$$ as a Label; there are several in last months listing, also.

Note especially the "Labels" in the $A000 thru $BFFF area; these are the locations where **EXTENDED BASIC** is using Routines within the BASIC's ROM. Though they are not **"DOCUMENTED"** locations for "External Access" by Radio Shack or Microsoft, they will have to remain firm or BOTH ROM's would have to be replaced during updates, etc. The TDP-100 uses the same ROM set, so Programs written using these locations would work on either machine. The other "similar" Computers (Dragon, etc.), DO use different ROM's; how about some of you readers in England, etc., letting us know how different!

Finally, there are **NUMEROUS JSR's** and **Internal JMP's** within BOTH the Extended and Regular BASIC ROMs. This means that **they are NOT Relocatable** without doing a lot of "Patching"; why they are written with a mixture of JSRs and BSRs, I have NO idea. I don't see that much difference in the amount of code required. Overall, the ROMs are extremely well written; I feel there was some reason that has escaped me. Again, some of you "more astute than I" (which does NOT require much astuteness) folks, how about a hint.

If there is enough request for it, we may publish the same thing for the Disk BASIC ROM; let's hear from you. The "project" now is **Documenting the Subroutine Entries.** A lot of you have a lot of these Chips figured out; send me the "Documentation" and I will pass it on. What we need is the Register Status upon Entry, what is accomplished, what is the Register Status upon exit, and which Registers are destroyed and which must be saved. This information will make it **MUCH EASIER** to write Assembly Language Programs for the Color Computer, because you can use the fundamental Routines that are already in the ROMs for much of your code.

(For those who missed last months "show", the " > " indicates a Subroutine Entry Location.)

```
L0000   8701
 L0001   844A 8907
 L0006   8795 8788 8812 8848 8838 8941 9780
 L0008   8885 88AF
 L000A   8384 8396
 L000F   8AA1 8AC7 8E68 8E82 9087 90A3
 L0010   9068
 L0019   8A91 8400 8B7B 9636 9662 96A7 96C1 96C6
         9804 9806 96E8 96FF 9745
 L001B   8986 898D 8843 884C 98A9 9684 9686 9688
         960F
 L001D   98D9
 L001F   86C1 86CF 98D7
 L0021   88E2 9680
 L0027   86E7
 L002B   8340 E98E 8998 89FF 8A04 8AC2 8B9F 930C
 L002C   8889
 L002F   82C2
 L0039   8894 9106 9240
 L003A   9069
 L003B   89ED
 L0040   838E 8910
 L0041   8884
 L0043   8849
 L0045   8891 90FB 9148 9249 9267
 L0047   8AC8 8AD9 8841 9029 902C 9040 9043 908B
         90CB 90CF 9009 90F7 9108 913E 9100 910E
         91E7 91FC 9263
 L004A   8382 83A0 8491 849F 8484
 L0048   8894 88AF
 L004D   8C4F
 L004F   839A 8398 8430 845B 84FB 8814 8935 9023
         9100 9102 9CD6 9CDA 9FED 9FF1
 L0050   880E 9225 9229
 L0051   8810 921F 9223
 L0052   879C 87EB 87EF 893F 8968 8882 88EC 8C4D
         8E59 8E42 9219  21D
 L0053   87ED 88EA 9213 9217
 L0054   8394 8380 852C 8816 8FF2 8FF6
 L0061   8498 84A2
 L0062   851C
 L0063   8814
 L0068   87DA 82E7 8894 8883
 L006C   86A9
 L006F   8273 927B 8288 8295 86AC 88FD 89E8 8C4B
         8C7E 9CF1
 L0070   8CC9 8CF9 8CFF
 L0072   8088
 L0078   8289 8344 8C6A
 L0079   8291 8CFB 800C
 L007A   8004 800A
 L007C   8346
 L007D   8351 8365
 L007E   834C 8355 9CL2 8CE6
 L008A   801A 82AC 833F 8D8A 8CD7 88C3 9088 99EC
         9A0F 9A22 90B6 9D83 9D92 90EA 90F4
 L0090   8CA7
> L009F   8154 92F1 931A 8322 86D8 87F2 860A 881F
         884C 885A 89DF 891D 8973 8960 8984 89C3
         8ACE 8B09 8B13 8C23 8C82 8C8C 8FC4 93EC
         9763 97B1 98F0 9EA1
> L00A5   8339 87EC 88E4 897C 8A16 8A1E 8A2B 8B17
         8B73 8C28 8C8C 8C3F 8ED2 8F96 8FB9 93F8
         994E 9984 9908 984C 967B 9786 9903 9EDC
         9FE3
 L00A6   82CD 8700 8398 8844  867 8871 8894 88C5
         88CA 88E7 8AE1 8AE8 8AF4 8B18 8B22 8B53
         9C30 9C36 9C3B
 L00AF   820E 86A9
 L00B0   8027A 8927
 L00B2   994C 999A 984A 98F9 9D52
 L00B3   9942 9957 995A 9646
 L00B4   98A2 98FF 9910
 L00B9   9380 95A7 990A
 L00BA   9292 932C 937C 9344 9497 956E 95D1 99EE
         9643 96F1 9789 97ED 0867 9E0F
 L00B7   9530 963A 965E 9666 96A0 96FD
 L00B9   9748 9754 975C 9840 98F3 988F
 L00BA   924D 92C9 9539 95E8 962E 9660 966A 96EE
 L00BC   965A 9698 96EC 974D
 L00BD   92FF 9310 9381 940C 9412 9418 9444 9446
         9458 9508 950A 9514 9518 951F 97D1 9844
         9981 99A5 99CB 98D2 98F0 9E0F 9F01
 L00BE   92B1 92B7 92CD 92D2 97C2 97C4
 L00BF   9395 941C 9432 9437 986E 9479 9484 95DD
         9511 9518 951F 952A 9712 970C 9E0A 9E13
         9F09
 L00CD   92AA 92C6 9958 99AC 9E09
 L00CE1  9353 950A 9618 9687
 L00C2   9366 93F3 959C 98FA 9CBC 9EC4 9EED
 L00C3   93A9 93D1 9410 9423 9440 971D 97CF 97D3
         97E3 97E7 97E9 97EE 9803 9855 9875 99BE
         99CB 99CD 9E17 9EFF 9F31 9F53 9F50 9F68
 L00C4   97C6 97C8 9808
 L00C9   9305 941A 9439 9477 9710 970A 970E 97F0
         97F4 9E19 9F03 9F41 9F72 9F7C 9FB7
 L00C6   9811E 98C7 9F45
 L00C7   9282 9389 9303 9DEC 9ED0 9E1F
 L00C9   8286 9 93 9307 9DF6 9E11 9E1D
 L00CB   99A1 99B7 99BA 9EAE 9F91 9F9B
 L00CD   99C8 9907 9CB2 9F70 9F7A
 L00CF   8A11 8A36 8A7B 977E 9823 9829 985F 9870
         9EBA 9FA7
 L00D1   8AD2 8A14 8A29 9776 9829 9B3B 9B7F 9946
         9970 9972 9EE9 9F35
 L0003   897A 8990 899F 89A6 8A3F 8A45 8A53 8A6B
         8C96 8CA5 8E45 8E69 8E84 8F9C 8FCA 8FCF
         9522 99F4 9F01 9F06
 L00D4   9784 978A 97F6
 L00D5   8A0F 8A1C 8A41 8A60 8CA4 8EAC 8ED0 8FCA
         97AE 98BC 9B2E 9C47 9CBE 9CF0 9E19 9E2B
         9F80 9F89
 L00D6   994F 9CFB 9E24 9E2D
 L00D7   8548 8501 85F1 8626 8842 8EA8 8EAE 8ED6
         8F83 8FB9 90F0 914A 9253 9256 925C 9270
         9278 945E 9463 9927 992C 9948 99A9 99C2
         98CA 98D2
 L00D8   853A 8558 8583 8A69 8A71 8F24 8F48 8F4F
         8FA3 904A 90B3 9102 9116 9134 975A 978F
         980C 9857 9884 990C 9A18 9A3  9A43 9A9F
         9B06 8B84 9BA4 9BD6 9BE2 9C12 9CD3 9CD0
         9D0C 9DA0 9EEF 9F89 9F91
```

```
> L0000  8E39 8E51 8E5B 8E6F 8E7B 8EBB BF1B BF22
         BF26 BF76 BF7E BFA1 9033 906D 9005 90EE
         911B 9142 9162 97B1 9890 991B 9A12 9A3D
         9B9F 9BA2 9BE4 9BEA 9C14 9C1B 9CD7 90A2
         9EF9 9F95
  L000A  8EB9 8EEB BF05 BF09 BF1A BF1E BF39 BF3D
         BF72 BF7B BF91 BF93 BFDC BFEA 9120 918E
         918A 9272
  L0008  938A 938C 9934 993D
  L000C  9931 9938 9931
  L000E  82A5 9A6D 9A6A 9B32
  L000F  829F 9A71 9AB8 9AD2 9AD8 9AE4 9B48 9B74
  L00E0  9ADA 9B51 9B7A
  L00E1  82AB 9ABF 9A96 9B29
  L00E2  82A3 9AB6 9ABD 9B2B
  L00E3  80C1 9B47 9B7D 9C41 9C49 9C4D
  L00E4  80C3 9C4F
  L00E5  82AA 9A9B 9AAF 9AD0 9ADC 9AE2 9B8E
  L00E6  8025 8C42 80F8
  L00E7  80C1
  L00E8  82AE 9050 9007
  L00E9  9065 90A9
  L00EF  9525 9E02
  L0100  90FC 9EC9
  L0100  9017
  L0112  801C 8955 895A 8964 8968
  L0110  8092
  L011E  9098
  L012A  8009
  L0137  8151
  L013C  816C
  L013E  8027
  L0167  808E
  L0168  8074
  L016A  8065
  L016B  806A
  L0176  8077
  L0177  8070
  L0179  8093
  L017A  8099
  L0188  8043
  L018C  8047
  L0191  805C
  L0192  8062
  L0194  8089
  L0195  806F
  L0197  804A
  L0198  8050
  L019A  8038
  L0198  803E
> L01A0  8162 8AFA 975C
  L01A3  8080
  L01A4  8086
  L0102  8023
  L010A  8CE8 8028
  L01E2  8C78
  L01E4  8C74
  L01E5  8336
  L01E7  8329
  L020C  89F2
  L020D  8555 8595 8604 8690
  L0300  9732
  L030B  8C12
  L03D9  88ED 88FA
  L03DA  9019 9062 9110 9126 9129 9145 9164
  L030B  8FE9 9185
  L4448  80A6
  L8031  8034
  L80C0  8088
  L800A  8009
  L80DE  8002
  L80E7  8082
  L8148  813E
  L8154  814A
  L8165  814E
  L8170  816A
  L817D  8174
  L81F0  8140
  L8257  817D
  L8273  8071
  L8285  8281 8289 828F 8293
  L8286  807A
> L829C  801F 808C
  L8289  803B
  L82BB  8302 8314 8318
  L82CF  8206
  L82D8  8203
  L82F1  82CA 82ED 8308
  L8304  9085
  L8310  8309 833B
  L8311  82F8
  L8316  82F7
> L831A  8316
  L834D  836A
  L835E  8357
  L8367  8362
> L836C  8327 832C 8334
> L837E  8386
> L83A3  8  3
> L83A6  8390
  L83AB  8378 83D1
  L83B8  8384
  L83C5  83BE
  L83D7  83CF
> L83DC  8386
  L83DF  830A
  L83E0  83C5
  L8410  8469
  L8432  8440
  L8437  8450
  L843C  846F
  L8441  847A
> L8446  8481
  L8489  8099
  L8491  848C
  L84AC  849A 84A7
  L84C4  84F2
  L84C9  8516
> L84F2  8489 848A
  L8501  8508
  L8504  84FF
> L8529  853E
  L852C  8527
  L8538  8578
  L8540  8588
  L855C  85AD
  L8590  856E
  L8570  8563
  L857D  8576
  L8581  855A
  L8594  89FA
  L858C  847F
  L8592  859F
  L8590  858E
  L85AB  85A3
> L85AF  85AB
> L89B4  8581 8590 85F3
  L85B6  85C0
  L85C2  8588 85C9
  L85C3  8581
  L85C7  85CE
> L8501  85C8 8608 8679
  L8505  85DB
  L85DE  8611
  L85F3  85E4
  L85F5  85E0 8607 8600 862C 8644
  L860F  85C9
  L8613  861C 8623
  L861E  861A
  L8625  85FE 8615 8653
  L8626  8602
  L862E  862A
  L8630  8632
  L8634  863A
  L8646  85E8
  L864A  8640
> L8650  8609 864A
> L8659  8620 8640 8673
  L865C  8648
  L8665  865E 8662
  L8668  8680 8683
  L8679  8671
  L867C  8677
  L8685  8660
> L8687  855D 85F9 8617 8667 868C 86A2
  L8694  8690
  L86A6  8696 869A 869E
  L86D8  8158
  L86EB  86E5
  L86FD  86E9
  L870E  8709
  L8724  8601 8703 8791
  L8727  8722
  L872E  872A
  L873F  873B
  L8746  8737
> L8748  8716 87A3
  L8768  8761
  L876B  8766
  L8776  8779
  L877B  8774
  L879C  878A
  L87BE  87E3
  L87CD  87D4
  L87D6  87B5
  L87D9  87B4 87C8
  L87DF  87D1
  L87E5  804D
  L87E8  8850
  L8800  880C
  L880A  87F6
  L880C  87FE
> L880E  8826 887A 8960 8884
  L881F  87FA 8832
  L882E  8821
> L8834  8808 8830 8850
  L8839  87DB
  L8845  87E7 886A
  L8846  8044
  L8862  8858
> L8866  887F 89C0
  L886E  88C3
> L88A1  887C 8884
> L88B1  888A 88B8
  L88B4  8854
  L88D9  8905
  L88EF  88F2
  L88F0  8856
  L890F  8878
> L891C  8911 892C
  L8927  891F
  L892C  885E
  L8943  893A
> L8944  8915 8962
  L894C  8014
  L8952  8946
  L8960  815E
  L898C  8986
  L8990  897E 898A
  L8993  8990
  L899F  8997
> L89AE  8533 8986
  L89B4  858B
> L89B8  89A1 8B3C 8B02
  L898F  8982
  L89C0  9380
  L89D2  89C7
  L89E1  89D4
> L89FC  8A1A 8A27 8A34
> L8A02  8A30 8A51
> L8A04  8A43
  L8A20  8A1B
  L8A2D  8A25
  L8A3A  8A32
  L8A3D  8A20 8A2D
> L8A67  8A49
> L8A68  8A57
  L8A71  8A81
  L8A72  8A73
  L8A83  8A38 8A47 8A70
> L8A86  8A6F 8A79 8A97 8AE5 8B85
  L8A90  8A8C
> L8A91  8A55 8A59
  L8A95  8A9F
  L8A99  8ABD 8AD1
  L8A98  8AAA
  L8AAC  8AC9
  L8AB2  8AA4
  L8AB9  8AB6
  L8ACD  8A8D
  L8AC7  8AD8
  L8AD3  8AA7
> L8ADD  8A48
  L8AE1  8AFD
  L8AE5  8ADF
  L8AE9  8AF8
  L8AEB  8811
  L8AED  8AF2 8807
  L8AEF  8819
  L8B13  8AFF 8803 8800 8877
  L8B17  8879
  L8B18  8A15
  L8B24  8829
  L8B41  8655
  L8B55  8833 8B3F
  L8B67  8861
  L8B71  8868
> L8B78  8A5B
  L8B7F  888E
  L8B8A  8895
  L8B8C  88CC 8807
  L8BAE  8B91
  L8B8E  88C7
  L8BC9  88C2
  L8BD8  8899
  L8BE5  8C0E
  L8BEA  88F0
  L8BFF  88F3 8AF8
  L8C07  8C03
  L8C08  8BF0
  L8C25  8C1F
  L8C42  8C39
  L8C44  8C2A 8C31
  L8C5F  8C57 8C88
> L8C62  8311
  L8C85  8C51
  L8C96  8C8E
  L8C9B  8C8D
  L8C81  8C8B
> L8C8F  8C9F 8CA3
> L8CC1  8C8F
  L8CC5  8CC8
> L8CC6  8C9B 8C81 8CC1
  L8CCD  8C87
> L8CDD  8C44
  L8COD  8C05
> L8CE2  8D10
> L8CE4  8C09
  L8CF1  8068
  L8D01  8CF5
  L8D02  8CFD
  L8D12  8D0E
> L8D14  8CDD
  L8D18  8D21 8D32 8D36 8D3C 8042
  L8D1D  8D19
  L8D26  8D2E
> L8D48  8D1B 8D89
> L8D58  8D1F 8D80
  L8D60  805C
> L8D62  8D30 8D99
  L8D6A  8D4E 8D66 8D60
> L8D68  8D40 8D80
> L8D72  8D34 8D3A 8D68 8D90 8DA7
> L8D7C  8CE8
  L8D89  8D8F 8D98 8D9F 8DA9 8D82
  L8D8B  8D87
  L8DA7  8DAE
> L8D8B  8D5A 8064
> L8D8C  8D72
  L8DC5  8DC7
  L8DC9  8DC8
  L8DD4  8DE0
  L8DDE  8DDA
> L8DE6  8DC5 8DE9
  L8DF6  8DEC 8DEF
> L8DF7  8DD4 8E10 8E12 8E17 8E28
> L8DF9  8DCD 8DF7
  L8DFD  8E00
> L8E04  8D28 8D93 8D97
> L8E0C  8D52 808A
  L8E10  8E2C
  L8E25  8E21
  L8E37  8EC1
  L8E38  8E75
  L8E9F  8E67
  L8E89  8EF5
  L8E71  8E80
  L8E82  8E79
  L8E88  8EF8 8F45
  L8C90  8056
  L8E95  8E97
```

```
  L8EA8  8E84
  L8EAE  8EA6
  L8EB4  8EAA
  L8EB7  8EB2
  L8EB9  8EBE 8F01
  L8EBB  8EED
  L8ED2  8F05
  L8EDA  8E41 8F98
  L8EDD  8ED8
  L8EE2  8ECA
  L8EEF  8EF4
  L8EF8  8F03
  L8F1A  8EF6
  L8F26  8F0B 8F13
  L8F24  8EC7
  L8F26  8F33 8F3F
  L8F41  8EF1
  L8F4F  8F2F 8F58
  L8F5A  8F37
  L8F74  8F20 8F52 8F5C 8F60 8F64 8F68 8F6C
  L8F8F  8F87
  L8F96  8F7C 8F81 8F8B
  L8F83  8E60
  L8FC4  8FB0
  L8FC6  8FB7 8FC2
> L8FDA  8E3C 8E88 8EC0 8EE6
  L8FE3  8FDE
> L8FE5  8FA8
  L8FF2  8FEE
  L8FFA  8FF4
  L9011  9015
  L9015  901C
  L9023  900C
  L902C  9027
  L9050  904C 910A
  L9054  9182
  L9060  9058 905C
  L9065  9071 9079 909C
  L907C  9092 9086 908A
  L9096  909A
  L909E  903E 9094 90A1
  L90A9  906F
> L90AA  8560 8B26 9092 98C2 9800
  L90B2  90AC
  L90B3  902E
  L90B8  90B5
  L90BD  90BA
  L90BF  90C9
  L90CB  90BF
  L90E2  90E6
> L90EA  9033 9051 900B 90F0 90F9
  L90EE  9003
  L90FF  90E0
  L9108  9104
  L9100  90C2
  L9116  9111
  L9118  9118
  L9129  9124
  L9130  913A
  L913C  9130
  L9141  913E
  L914A  9156
  L9167  9190
  L916F  9164
  L9177  917A
> L9185  9054
  L919E  9194 919A
  L91A0  91AC 91B0
  L91B6  91A4 91A6
  L91BA  91B4
  L91C4  91BE
  L91CC  91C8
> L91C0  9029 9115
  L91E0  91E2
  L91E4  91EF
  L91E9  91D6
  L91F1  91FE
  L9200  9157
> L9202  9030 90FF 914C
  L9214  9245
  L9213  922E
  L9235  9230
> L9249  9047 9211 9270
> L9240  90F2
  L9256  924B
  L9262  925B
> L9263  903A 9000
  L926A  926C
  L927A  9276
  L9278  928C
> L92B1  9042 9151
  L92BE  9282
> L928F  9298 940F 9918
> L9298  933F 9374 9455 947F 9805 9B64
  L929C  928F
  L92B0  9288
  L92E4  9206
> L92E9  9490 9B18
> L92E0  92F5 9883
  L92F3  92EE
> L92FC  931A 9385
> L9303  93A0
  L9309  9305
  L9317  9312
> L931A  936B
> L9310  933C 9420 98F5 9EA9
> L9320  9426 952F 9EBF
  L932C  9324
  L9338  9330
  L9349  934F
  L9351  9348 935F
  L9356  939C
  L9358  9347
  L9366  9563
> L9377  9460 94E4 99FE
> L938F  93CE 9765
  L939E  9399
> L9382  9339 9598 98F2 9EA6
  L93B8  9380
  L93CE  93C3 93C7
  L93E9  930E
> L9420  93EE 97EB 9E21
  L9429  9406
  L9430  9450
  L9432  9441
  L9434  942C
  L9443  943B
> L9444  9408 941E 9454 94AB
  L9451  944B
  L945E  9469 946B
  L946C  94BA
> L946E  940A 9414
  L947B  9479
  L948A  9494
> L9404  945C
> L9490  9486
> L94A1  93FA 9E28 9F0F
  L9482  94AC
  L94C1  94BC
  L94CC  94C6
  L94DB  94D2 9408
  L94E2  94FA 9500
  L9502  94E9
> L9506  94B4 9976 998C 99EB
  L9500  94A1
  L930E  94D4
> L9514  94BE 9983 99E2
  L951B  94AE
> L9522  9915 9EA3
  L9536  9544
  L9538  955F
  L9542  9532
  L9552  9548
  L9559  9550
> L955A  9534 954A 9555 9593
> L955D  904F
  L956C  9566
  L9576  957C
  L9578  9574
  L9579  957F
  L9578  9571
> L95B1  936E 98FC 9907 9EC6
  L9598  9586 958A 9991
> L959A  93F5 9581 9CC0 9D54
> L95A2  9596 959E
> L95AA  9678
  L95AC  B279
  L95CF  95AA
  L95F7  95F4
> L95FB  95F7
  L9600  960C 9614
  L9607  9601
  L9609  9605
> L960F  95EB
  L9616  9689
  L9650  9623
  L966C  964E 9670
  L9660  962A 9638 9654 9664 96BF 9693 9r
  L967F  9672
  L96B7  9683
  L96BD  96C4
  L96C8  96E4
  L96D4  96BB
  L9608  96E2
> L96E6  8098
  L9707  9631
> L9710  9472 94A5 9705
  L9714  9721
> L9710  9448 94B7 97CA
  L9736  973C
> L9738  9723 972C
  L9751  9714
  L9752  9743 9747 9782
  L975A  9756
  L9765  9761
  L979A  9791
  L979F  97A9
  L97AE  97A6
  L97B7  97BB
  L97C0  97BC
  L97CA  9798 9785
  L97D3  97C0
  L97DE  9708
  L97E8  97E5
  L9808  9820
  L980C  9817
  L9816  9857
  L9822  982C
> L9823  9810 9835
  L982E  9802
  L9831  980E
  L9839  979C
  L9852  97F8
  L985D  9862
  L9AA4  9859
  L986E  986A
  L9875  98C9
  L9877  9888
  L9884  987A
  L9890  98BA
  L98A7  98B6
  L98B1  9899 989F 98A5 98A9
> L98CC  9768
  L98D7  98E6
  L98E8  98E0
  L98EB  994A
  L98F7  98EE
  L990A  9905
  L9951  9920
  L9934  9954 999E
  L993B  9936
  L9994  994C
  L995B  9951
  L9969  9963
  L996C  998A
  L996E  9958
  L9970  9981
  L9983  997C
  L998C  9974
  L999E  9996
> L99A1  9967 999C
  L99AC  99C4
> L99BA  9929 992E 996C
> L99CE  99B6 99E0 990F
> L99CB  9922 9969
> L99DF  9910 995A
> L99EB  9988 990C
  L99EE  99E6
  L99F2  9A07
  L9A09  99FC
  L9A0B  99e2 99r8
> L9A12  9979 99r4
  L9A32  9C18
  L9A37  9A3B 9A45
  L9A39  9A35
  L9A43  9A4C 9A50 9A5A
> L9A5C  9A58
  L9A60  9A5L
  L9A0B  9A6F
  L9A9A  9A9C
> L9A9F  9A9A
  L9AA0  9AA1
  L9AAF  9AAB
  L9AB2  9ABD
> L9AC0  9A63 9A77 9A91 9AB6
  L9AC3  9AB4
  L9ACD  9A68 9A7B 9A94 9ABB 9B25
  L9AD0  9AC8
> L9AE7  9ADE
  L9AEB  9AC5
  L9AF2  9AED
  L9AFA  9AF4
  L9AFF  9AFB
  L9B15  9B0F
  L9B1B  9B13
  L9B1F  9B1A
  L9B22  9AF0 9B08 9B16 9B10
  L9B2B  9AE9
  L9B49  9B55
> L9B57  9B49 9B4F
> L9B5A  9B40 9B53
  L9B5F  9B61
```

```
  L9864   9836
  L9872   987E
> L988D   9872 9878
> L9885   9876 987C
  L9888   9889
> L988C   9845 986E
  L9897   9890
> L9898   9A47 9AA3 9AEF 9B0A 9BAC 9BDA 9C1F 9C27
          9CCB 9CE1 9D10 9E32 9E3C 9E58
  L9894   9BA8
  L9BAC   9AC0
> L98BE   9AFA 9CCE
  L98C8   9BED
> L98E2   9AAA 9B1F 9D18 9E66
  L9BFB   989C 98C5 9BD0 9BD4 98EF 98F3 9BFB 9BFD
          9C29
  L98EE   9B9D
  L98F1   9BDB
  L98F2   9BB4
  L98F7   988C
  L98FC   9B88
  L9C01   98CD
  L9C04   9A54
> L9C1B   9C03 9C0A 9D9B
  L9C27   9C2C
  L9C3E   B95D
  L9C5A   9C48
  L9C5B   9B01
  L9C62   9838
  L9C7A   9864
  L9CC6   90A6
> L9CC9   9AC7 9D1F 9E6C
  L9CD1   9C05 9CDF
  L9CD3   9CC9
  L9CDD   9CE6 9CEA 9CF2 9CFA 9D57 9E2F
  L9CF4   9CEE
  L9CFC   9CF6
  L9D21   9DBE 9D1D
  L9D4C   9D58 9D63 9E41
  L9D4F   9D25
  L9D57   9D5F 9D67
  L9D59   9D29
  L9D61   9D2D
  L9D69   9D44
  L9D6D   9D44
  L9D72   9D41
  L9D7B   9D47
  L9D82   9D3D
  L9D83   9D84
  L9D87   9D39
  L9D8C   9D35
  L9D8F   9D31
  L9D92   9D80
  L9D98   9CFE
> L9DA9   9D2D 9DD3
  L9DB6   9DB2
  L9DB0   9D6C
> L9DC3   9118 9D84 9DE2
> L9DC4   9D6A 9D75 9D7E 9D8B 9D90
  L9DC7   9CD9 9DC1
  L9DCB   9DAB
  L9DCB   9D7D 9D79 9D80 9D85 9D96 9E5B
  L9DDC   9DE6
  L9DEB   9DDE
  L9DF2   9DEE
  L9DFC   9DFB 9E54
  L9E05   9EDD
  L9E0D   9EDB
  L9E21   9E1B
  L9E2B   9E26
  L9E32   9DD4
  L9E36   9E3D
> L9E5B   9EA5
> L9E5E   9E37
  L9E69   9E60
  L9E6A   9E64
  L9E7B   9E77
  L9E7B   9E07
  L9EA3   9E9F
  L9EDF   9ECE
  L9EE9   9EE3
  L9EFD   9FA3
  L9F11   9F0C
  L9F51   9F29 9F2D
  L9F5B   9F4B 9F4F
  L9F66   9F9F
  L9F6B   9F55 9F59 9F64
  L9F1A   9F6E
  L9F89   9F7E
  L9F87   9F74 9F7B 9F83
  L9F8F   9F8B
  L9F9A   9F93
  L9FA3   9F9D
  L9FA6   9F9B
> L9FA7   9F37 9F2D
  L9FB4   9FAB
> L9FB5   9D86 9F37 9FBD
  L9FD4   9F0D
> L9FDF   9FBD
> L9FE2   9EF1 9EF6
  L9FF8   9FE5
  LA0E2   B080
  LA0E8   B0CD
> LA171   B687
> LA176   BCC6
> LA282   82E4 82EE 859A 8659 BE8B BEC5 BFE0
> LA35F   8686
> LA3ED   89CC
> LA406   8683
> LA429   8C18
  LA42D   8CAD
  LA444   8299
  LA44C   851E
  LA491   835B
  LA498   8C66
  LA5D5   8C77
> LA578   8324 8C25 8C6E
> LA59A   80DA 8743
> LA5A5   89C9
> LA5AE   86R0
> LA5C7   6981 8A3A BC53 BC98
  LA5EA   86BB
  LA616   8C5F
  LA619   BCCD 8D59
> LA635   8CB0
  LA64A   8CEE
> LA64B   8C71
> LA65F   8341
> LA7D9   8348
> LA7E9   88F4
> LA7F4   8367
> LA974   88F7 9A3F
> LA97A   9A2F
> LA947   9A2C
  LA98B   9C43
> LACIE   8B4E
> LAC33   99B4 9CDF 9D9D
  LAC46   8B6E BCDF
  LAC6D   8988
  LAC75   89A6 89AB AA64
  LAC7C   8C9C
  LAC9D   83D6
  LACA8   859A
> LACEF   844E 8A61 98CB
> LACF1   89AB
> LAD01   853C 8977 8AD6 84C4
> LAD19   8CA3 9703
> LAD21   89A1
> LAD26   8409 8A5E 96CE
> LAD33   8FEA
  LAD9E   96D1
> LADC6   82FF
  LADD4   8145
> LADEB   82B0
  LAE15   8701
  LAED2   853F
  LAEED   8F9E
> LAF67   8914 89AE 89FC 8B1F
  LAF44   89F9
> LB035   89E3
> LB141   8BD9 8B9E 9ED3 9FEA
  LB1A3   8881
> LB146   8799 89EF 8E4A 8E9A
> LB156   874B 8759 8781 8796 8E47 9A28 9CE3
> LB158   8E97
> LB244   8906
> LB762   8178 8888 8932
> LB767   852B 8710 875C 87A8 8B8C 9371 9388
> LB26A   86BE 86D8 874E 877E 8880 9368 93A3 9382
> LB260   8362 8756 8793 87A0 89CF 8A22 8A2F 8C2C
          8C90 8F8F 9309 93FE 9552 958C 9768 978C
          9E84 9ED0 9FE7
> LB26F   8715 8891 88A5 8946 890B 8E9F 93AD 93CB
          9403 942B 9729 9795
  LB277   8090 8165 82CC 88D2 88E8 93E2 97A8
> LB2B4   9C36
  LB2CE   8180
> LB347   86C9 86D8 8887 89EA 98CC
> LB35C   8B4C
> LB3A2   8425 9C22 9D13
  LB4AA   8D2C 8330 8449 8724 897D 8A83 8C3E 8E84
          8F47 955F 966D 96A3 9752 982E 9809 9ACD
          98E8 9DAC 9F3D
  LB4F3   870C 9356
  LB4F4   8623
> LB50F   876F
  LB518   8C19
> LB51A   89F6
> LB56D   85E2
> LB643   E5F4
> LB654   874B 9A32 9CC6
> LB697   B93C
> LB659   87A3 8E0F
  LB698   8778
> LB644   8768
> LB64D   8653
> LB7DB   8791 8C33 955A 9625 9674 96B8 973F
> LB7DE   8153 878C 9CD5 9FF3
> LB734   92FC 93A6
> LB73B   86FD 8708 9650 967F
> LB73F   836F 8949 8C93 9EB1
> LB740   8F2D 9EDC
> LB7C2   8543
> LB998   8378 8585 8592 85A5 8BA9 8ED4
> LB95C   88FF
> LB99C   8085 8B9C 8F80
> LB99F   89DE 8E56
> LB9AC   8552 8E64
> LB9AF   8902
> LB9B4   9204
> LB9B9   83D4 8466 8515
> LB9C2   8378 845A 8472
  LBA1C   881C
  LBA3A   848E
  LBA92   839C 8836
  LBAC5   83CD 8463
> LBACA   847D 84B7 84F5 910B
> LBB4B   8520
  LBB5C   8501
> LBB6A   91F9
> LBB82   90C3 9134 91E4
> LBB8F   83A3 8456
> LBC14   8391 8456
> LBC2F   8383 84F8
> LBC35   8388 8494 88D6
> LBC4C   84AC
> LBC5F   8480
> LBC6D   8446 8524
> LBCAC   84A4 90C9 91EC 91F4
> LBCCB   9207
> LBCEE   849C 85CA 8520
> LBD99   8477
  LBDB6   91F1
  LBDBF   91F9
  LBDC9   90D6 91D8
> LBDC5   88A6
> LBDCC   82E9 854F 88A3
> LBDC9   88B7 9C0E
  LBEC0   8483
  LBEC4   920C
  LBEE9   830C 848F 8530 9747
> LBEF0   83C8 846C
> LBFFF   8519
  LBF18   837E
  LBFA6   83A8
  LCDC2   9DAC
  LFFC6   9611
  LFFCB   954E
  LFFDD   80D7
```

"C" User Notes

Norm Cowmo
3 Pryor Road
Natick, MA 01760

This month I would like to go over some string functions that many of you might not have in your libraries. People who do major software projects inevitably find themselves manipulating stings in some form or another. A rich library of string manipulators is certainly an asset in those times.

Most good C libraries will have a few of the basic string operators. These usually include

```
strlen(s)       return the length of s
strcpy(t,f)     copy f into t
strcat(t,f)     add f to the tail of t
strcmp(s1,s2)   compare s1 against s2
```

it has generally evolved that the string which is acted upon is passed as the first argument. Think of them as "to" and "from" or "destination" and "source".

There are several others that are also very handy. Most libraries include some but not all of them. A few I have not found in the libraries that came with any of the 6809 C compilers I've tested to date. These functions include

```
strclr(s,n)       clear n bytes of s
streq(s1,s2)      test if s1 and s2 are equal
strncpy(t,f,n)    copy n bytes from f to t
strncmp(s1,s2,n)  compare n bytes of s1 and s2
strneq(s1,s2,n)   test n bytes of s1 and s2
reverse(s)        reverse the characters in s
strins(t,f,n)     insert f into t at n
strdel(s,n,o)     delete n bytes from s at o
strsub(s1,s2)     find the position of s1 in s2
```

strclr() is straight forward and is nice to have in your library. While most string handlers will leave a string terminated, you may occasionally need to insure it with this function.

```
/*
 * clear cnt characters of s, return
 * a pointer to s
 */
strclr(s,cnt)
    char *s;
    int cnt;
    {
    char *pnt;

    pnt = s;
    while(cnt--)
        *pnt++ = '\0';
    return(s);
    }
```

Almost as simple is streq(). All it does is invert the logical meaning of the value returned by strcmp(). Since strcmp() returns the difference between the nonmatching characters, a zero value implies that the strings were equal. This is sort of counter to how C usually interprets zero (as FALSE) so streq() turns it around to make the program a little more readable.

```
/*
 * test s1 and s2 for equality
 */
streq(s1,s2)
    char *s1, *s2;
    {
    if (strcmp(s1,s2))
        return(FALSE);
    return(TRUE);
    }
```

strncpy() allows you to limit the number of characters actually copied into s1. It might be used to insure a uniform string size.

```
/*
 * copy no more than n characters
 * of t to f
 */
strncpy(to,from,cnt)
    char *to, *from;
    int cnt;
    {
    char *pnt;

    pnt = to;
    while((*pnt++ = *from++)  &&  --cnt)
        ;
    *pnt = '\0';
    return(to);
    }
```

The lone semicolon after the while statement is no mistake. All the work is done in the conditional test. Copying the NULL into s1 at the end is an escape hatch for the case where the loop is exited with n == 0.

This example brings out something subtle. At first glance, the following two fragments may appear to have synonymous function.

```
while (*s1++ = *s2++)
    ;

while (*s2)
    *s1++ = *s2++;
```

They are not the same. The first example would copy all of s2 into s1 including the terminal NULL character. This happens because the character is copied, and the pointers updated before the comparison is made. In the second case, by testing for the end of s2 first, the terminal NULL is not copied.

strncmp() is similar in that it lets you put a limit on how many characters are actually compared.

```
/*
 * compare no more than n characters
 * between s1 and s2
 */
strncmp(s1,s2,cnt)
    char *s1,*s2;
    int cnt;
    {
    while ((*s1 == *s2)  &&  --cnt)
        {
        s1++;
        s2++;
        }
    return(*s2-*s1);
    }
```

strneq(s1,s2,n) is similar to streq() except that strncmp() is called, and n is passed along also.

```
/*
 * test no more than cnt characters of
 * s1 and s2 for equality
 */
strneq(s1,s2,cnt)
    char *s1, *s2;
    int cnt;
    {
    if (strncmp(s1,s2,cnt))
        return(FALSE);
    return(TRUE);
    }
```

Why reverse() never ended up being called strrev() is beyond me. I guess the best of conventions break down at times. At any rate, reverse() takes a string and reverses the order of all the characters with the exception of NULL which remains at the end.

```
/*
 * reverse the character sequence
 * in s, return pointer to s
 */
reverse(s)
    char *s;
    {
    char *hd, *tl, temp;

    hd = s;
    tl = hd + strlen(hd) - 1;
    while (hd < tl)
        {
        temp = *hd;
        *hd++ = *tl;
        *tl-- = temp;
        }
    return(s);
    }
```

The next functions I have not really seen in may C libraries. I got the bug to code them after reading an article in one of the ACM (Association of Computing Machinery) Journals. The article said that they were useful Pascal functions and procedures. Well if they're useful, then they're useful in any languege and that includes C.

strins() inserts one string into another at a given offset. The first character in the string is assumed to have an offset of zero.

```
/*
 * insert from into to at offset, the
 * first position is zero.
 */
strins(to, from, offset)
    char *to, *from;
    int offset;
    {
    int i, j, len1, len2;

    /* split up the destination */
    len1 = strlen(to);
    len2 = strlen(from);
    i = len1;
    j = len1 + len2;
    while (i >= offset)
        to[j--] = to[i--];

    /* splice in the source */
    i = 0;
    j = offset;
    while (i < len2)
        to[j++] = from[i++];
    return(to);
    }
```

strdel() deletes a portion of a string. Note that the first character is assumes to have an offset of zero.

```
/*
 * delete cnt characters from s starting
 * at the offset
 */
strdel(s,cnt,offset)
    char *s;
    int cnt, offset;
    {
    int len;

    len = strlen(s);
    if (offset + cnt >= len)
        return(s);
    strcpy((s + offset), (s + offset + cnt));
    return(s);
    }
```

strsub() returns the position of the first occurrence of one string within another string. If the string was not found, it returns an error code of -1.

```
/*
 * return the position of the first occurance
 * of s2 in s1, or -1 if s2 is not found.
 */
strsub(s1, s2)
    char *s1, *s2;
    {
    int cnt, len1, len2;
    char *pnt1, *pnt2;

    len1 = strlen(s1);
    len2 = strlen(s2);
    if (len1 < len2)
        return(-1);

    cnt = 0;
    len1 -= len2;
    while (len1--)
        {
        pnt1 = s1++;
        pnt2 = s2;
        while (*pnt1 == *pnt2)
            {
            pnt1++;
            pnt2++;
            }
        if (*pnt2 == '\0')
            return(cnt);
        cnt++;
        }
    return(-1);
    }
```

These functions have all been tested and work. For the most part they should be pretty bullet proof. I tried to anticipate and handle the common error modes, but you might want to think through them yourselves just to be sure. There is also the option of changing the number parameters from being "zero based" to begin "one based". It's your choice.

I originally wrote and tested most of these functions using for loops instead of while's. I then tried to compile them with the Word's Worth compiler. In most of them, I had two variables being initialized and incremented. It was strictly a no-go situation. I had either uncovered a bug or the fact that the compile doesn't implement the complete for loop syntax. I have not talked with them about it so I won't speculate.

Just to make sure that it wasn't some stupid little thing that I had over looked, I coded up a quick little test case as follows

```
main()
    {
    int i, j;

    /* case #1 */
    for (i = 0; i < 10; i++)
        j += 10;

    /* case #2 */
    for (i = j = 0; i < 10; i++, j++)
        j += 10;

    /* case #3 */
    for (i = 0, j = 0; i < 10; i++, j++)
        j += 10;
    }
```

Case #1 compiled with flying colors. Case #2 failed. The compiler expected either a semicolon or bracket (I have forgotten) in place of the comma following i++. Case #3 failed because the compiler expected a semicolon in place of the comma following i=0. It also failed again after the i++.

I just wanted to let you know in case some of you were encountering these errors and thought that it might be some problem on your part. It sure had me fooled; to the point that I even got out the master copy of version 2.02 as a check.

WHAT'S NEW

Simply put, very little. I haven't yet received the Microware compiler; but I expect it shortly as I understand that they are now (mid-March) testing it in-house. I was so busy this month getting OS9 level 1.2 up on my system that I haven't had a chance to really try out the introl floating point package yet.

By the end of summer I hope to have another system up and running. This one will even be quasi-commercial. It will be built up on the SS50 bus and have 8 inch drives. I am toying with the idea of getting and auto-answer modem and another phone line. That would open up the possibility of letting you dial up and down load source code that appears in the column or depositing comments and opinions.

Next month's column is still quite embryonic. That's a euphemism for the fact that I haven't got a clue for what's next. Till then...

- - -

'C' FLOATING POINT

By Allan R. Batteiger
and Howard L. Harkness

Floating point math was one of the features originally omitted from the Word's Worth Middle-C(tm) distribution package. This was primarily due to the structure of the original 8080 compiler written by Ron Cain, which would require an almost complete re-write to add a floating point type and the associated intrinsic functions. However, it is still possible to get the use of floating point even though it is not actually part of the compiler.

This article is the first of a series describing a floating point library written especially for Middle-C. The concept of the external floating point library was borrowed from the BDS C compiler, which is one of the most widely distributed C compilers for the 8080/Z80 crowd (mob). The code is adapted from a similar package originally written for the 6800, modified to run on the 6809, interface with Middle-C, and emulate the AMD9511.

The 9511 format was chosen because it is available on several 6809 systems (and because Allan has one in his system to test the package against!). Users who are thinking of upgrading their systems to use a 9511 (if it ever gets cheap enough) can get some experience with it ahead of time. Then, when the happy event eventually does happen, very little, if any, code will need to be changed in the applications packages to make them work with the 9511. Although the package was designed for Middle-C, it will probably run with any currently available 6809 C compiler with only minor hassles.

This packagage includes the four basic functions, along with utilities to normalize, complement, convert, input, and output floating point numbers. Since the package is not 100% complete as of this writing, we may add other functions that we think of along the way (suggestions welcome).

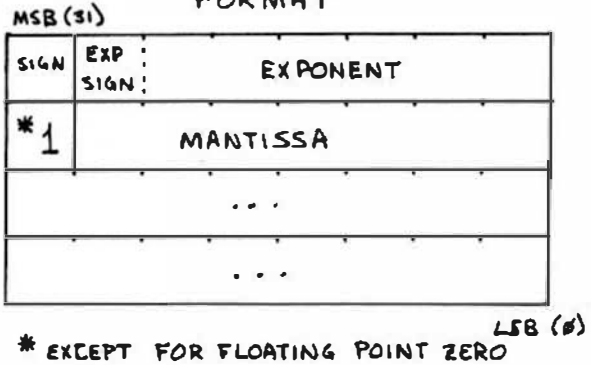
FLOATING POINT OVERVIEW

Floating point requires two parts. The first is called the mantissa, which is the 'significant digits' of the number. The second is the exponent. This does not represent a power of the mantissa, but refers to the power of two by which the mantissa will be multiplied to calculate the actual value of the number. This corresponds to the number of bits that the mantissa had to be shifted before it was 'normalized', or left-justified in the floating point representation. To do floating point math where there is no type 'float', we must use an array of some other available type large enough to hold the representation we have chosen. For single precision in the 9511, this requires 32 bits, or 4 bytes, which can be either four characters (char float[4];) or two integers (int float[2];).

The mantissa is expressed as a 24 bit fractional value. The exponent is an unbiased 2's complement 7-bit value with a range of -64 to +63. 'Unbiased' means that there is no constant added or subtracted to the exponent before the value is determined (biased exponents are fairly common in floating point hardware). The most significant bit represents the sign of the mantissa. Bit 23 must be 1 except when the value of the floating point number is zero, in which case all bits in the number are zero. The total dynamic range possible with this format is +/- 2.7 X 10**-20 to +/- 9.2 X 10**18. This should be suitable for the vast majority of floating point applications.

The core of the 9511 emulator is written is 6809 assembly, with the rest in C. In our next installment, we will present the source code for the core portions (and as much more as will fit into a reasonably-sized article), and a more detailed explanation of the theory of operation. By the time you finish this series, you should be able to glibly rattle off the buzzwords associated with floating point math, and may even be able to (patiently) explain the whole thing to your uninitiated friends.

AMD 9511
FORMAT



* EXCEPT FOR FLOATING POINT ZERO

HOW TO GET THIS PACKAGE

Current registered owners of Middle-C will be recieving the floating point package with their free 2.1 update. Note to customers: If you haven't already done so, please send your Middle-C 2.0, 2.01, or 2.02 disks back for the update now. For your convenience, you may send a copy of the invoice and $6.00 ($7.50 overseas) in lieu of the original disk.

If you are not a current Middle-C owner, you may purchase Middle-C 2.02 for $99 now, or 2.1 for $110 when it is announced. If you have another compiler already, and only want the floating point package, send $17.50 to Word's Worth, PO Box 28954, Dallas, Texas 75228. However, the published source will probably be short enough that typing it in will not be terribly onerous.

OS9 USER NOTES

By: Peter Dibble
517 Goler House
Rochester, NY 14620

I just installed OS-9 Level Two Version 1.1. Finally it's not "preliminary" any more. Since OS-9 never was very unreliable it is hard to tell whether it is more reliable, but it is very easy to appreciate the new utilities. I spent months writing a PWD program. It prints the name of the current data or execution directory. I hoped someday maybe I could sell that program. Well, Microware beat me to it. The new versions of OS-9 include PWD and PXD, Print Working Directory and Print eXecution Directory. They also added a DELDIR command which deletes a directory with all the files in it, a command called IDENT which displays information about modules in files, a file comparison utility called CMP, and two commands called BINEX and EXBIN which convert a file to and from Motorola standard S-Record format. DCHECK, the program which checks disk structure, now seems to work correctly, and DSAVE, the command which constructs a procedure file to copy groups of files, has been substantially enhanced, but Level Two users will have to continue to live with numeric error messages. A command called PRINTERR, which is supposed to instruct the operating system to use text error messages, wasn't on my distribution disk.

An important new feature in OS-9 is support for XON/XOFF. The ASCII character set includes 32 special codes such as backspace ($08) and escape ($1B) which don't generally represent printable characters, but still have defined meanings. XON and XOFF are among the more useful of these special codes. If, for instance, you have a terminal which usually runs at 19.2KB, but can only accept input at about 200 characters per second when it is in insert mode, it would be nice to be able to constantly adjust the speed at which the computer is transmitting to match the speed at which the terminal can receive. In general you can't do that, but often it is sufficient to be able to tell the computer to "hold it," and "go ahead." If the computer can deal with XON/XOFF protocol, it will "hold it" whenever it receives an XOFF, and "go ahead" whenever it receives an XON. There are quite a few terminals and printers around which run much better when they are attached to a computer which supports XON/XOFF. It is interesting to note that XOFF (often called OC3) is entered as <CTRL>S, and XON (DC1) is <CTRL>Q. In order to use this protocol you've got to find some character other than <CTRL>Q to use as the "quit" character. I wonder whether Frank Hogg is going to be able to adjust DynaStar so it can live without <CTRL>Q and <CTRL>S.

One of the first things I do with a new version of OS-9 is put together a new bootstrap. There is nothing really wrong with the bootstrap that comes with the system, but I have my own Device Descriptors and Drivers, and even if I didn't need to, I probably would want to re-generate the bootstrap just on the principle of the thing. The modules in the bootstrap are automatically loaded when the system is booted, packed efficiently into memory, and made permanent. It sounds as though, if you have enough memory, it would be a good idea to include in the bootstrap file all the modules you would like permanently in memory. Don't do it! Modules in the boot file are not only permanently in storage, they are also permanently attached to the other programs in the boot. Say you put a P-Code interpreter in the bootstrap - when you link to that module in order to use it, you drag everything else in the bootstrap along with it. If you have a 48K bootstrap you would only be able to run programs which use up to about 12K total. Modules you expect to link to should not be included in the bootstrap. If you include a utility command such as COPY, you may find that you can only use a relatively small amount of memory with COPY. The best way to handle commonly used commands is to merge just less than some small multiple of 4K of them into a utilities file and load it using a LOAD command in the startup file. Since my system allocates memory in blocks of 4K, small programs like COPY and PWD only waste memory if they are loaded by themselves. By collecting groups of programs together you use memory more efficiently, essentially keeping two or more programs in the space normally allocated to one. If your version of OS-9 allocates memory in different sized hunks, the size of the group of programs should be changed to reflect the new constraints. Users of Level One systems don't have to worry about any of this stuff.

The first time I generated a new bootstrap was a little bit intimidating. It is important to realize that, provided you are marginally careful (don't spill chocolate milk on an important disk, etc.), the worst you can do is waste your time. If you don't have a lot of memory the chance to remove unused device descriptors from the bootstrap may be worth the trouble involved in running OS9GEN. If you want to change any modules which are in the bootstrap (addresses in Device Descriptors for instance), the cleanest way to do it is to modify them them with DEBUG, save the modified modules, fix their CRC with VERIFY, and build a new bootstrap with the modified modules. A module must be saved on disk in order to be included in the bootstrap. You should use the SAVE command to create files containing each module you might want in the new bootstrap. Build a file with the names of those files you want to combine into the new bootstrap, and use that list of files as input to OS9GEN. Finally use DCOPY to copy all the other files on your system disk over to the new one.

I have many files on my system disk that are not part of the OS-9 operating system. An important part of installing a new version of OS-9 which is not mentioned in the manuals is copying all the

non-OS-9 files you need onto your new system disk. I have discovered an easy way to do this. I imagine most of you OS-9 users already know this trick, but I wish someone had told me about it a year ago. By running DSAVE on your old system disk you can create a file containing a copy command for each of the files on your old system disk. If you add a "-x" as one of the first few lines in that file it won't quit if one of the commands fails. The copy commands for files that are already on the new disk will fail, but the procedure will proceed to the next command instead of quitting. The result is a disk with all the files you want on it.

Most of the programming I do is on machines with far more than 64K available to each program. It is easy to get used to having effectively unlimited memory. The 6809 can only use 64K, but with the help of OS-9 Level Two (not Level One) it is possible to use more memory than most people can afford. Over the next few months I expect to spend some time discussing various ways of doing this.

One of the basic facilities in OS-9 (and most other sophisticated operating systems) is called FORK. The effect of FORK is to set a program up and start it running without interfering with the program which FORKed it. Each FORKed program is called a Process or a Task. A process can run for all practical purposes at the same time as the program that FORKed it. Part of setting a process up is finding enough memory for it to run. In OS-9 Level Two each process runs in its own "address space"... that is, no user process shares any memory with any other process except by special arrangement. If you have enough memory, each process can occupy all of its 64K address space except a shred reserved for OS-9.

I have been spending a lot of time writing a program which I call a "smart terminal" program. It started out as a program to allow me to communicate with a variety of computers without having to unhook my terminal from my computer, and fuss with half/full duplex. It just keeps growing. One thing I decided to do was include a way of printing a screen full of data. You can't just stop everything and print the screen; it would take so long to print that the input buffer from the modem would overflow, and at best data would be lost. A solution is to use a FORKed process to print the screen. Once I realized that I could start a process to print the screen, I carried it a step farther and fixed things so I can ask to have lots of screens printed, start a process for each screen, and let them queue up for a chance at the printer while the process doing the smart terminal bit runs cheerfully along. At about 8K per process (4K for the module and 4K for variable storage) the minimum allocation on my Level Two system, I can queue up about 20 screens in the 200K I usually have available. Using the more efficient allocation of storage available under Level One I could probably have queued up about 10 screens in a 56K system. I admit this is a trivial example of the use of extended storage, but the point is that this is a simple example of the kind of thing you can do with extended storage. It is easiest to use multiple processes to get at lots of storage when you can spin off a task that can run in isolation. Communicating between processes is another problem, suitable for investigation in at least one future column.

MICROWARE USER SEMINAR

Second Annual Microware User Seminar
August 12-15, 1983

This is an event you won't want to miss if you use, sell or are interested in Systems that use Microware 6809 software. Last year's Microware User Seminar was an outstanding success, and the 1983 edition promises to be even bigger and better. The 4-day event will be held August 12 through August 15 at the Des Moines Marriott Hotel.

There will be informative roundtable technical sessions covering almost every aspect of the design and use of Microware software. At these sessions you can pose questions to the Microware staff members who designed each program. Many attendees of last year's seminar remarked that the opportunity to meet and trade notes with other 6809 user made the seminar a very important experience for them.

You will also get a preview of Microware's exciting new 68000 software products plus demonstrations of new 6809 software.

A bigger and better Exhibit Area will have display booths from many leading suppliers of OS-9 compatible hardware and software. Their representatives will be available to answer your questions. This will be a golden opportunity to see and compare what's new in computer systems, disk and storage devices, and application software for OS-9.

A buffet dinner is planned for Friday, August 12th, and a brunch will be held the morning of August 14th. Guest speakers will be featured at both events. Part of the evening of August 13th has been set aside for a meeting of the OS-9 User Group.

Don't miss this chance to increase your knowledge and skill in the latest in microcomputer software technology - register for the Microware User Seminar today!

GENERAL INFORMATION ABOUT THE
SECOND ANNUAL MICROWARE USER SEMINAR

SCHEDULE OF EVENTS:

FRIDAY, AUGUST 12
Registration: Noon to 5:00 PM
Exhibit Hall opens at Noon
Cocktail Reception: 6:30 PM
Buffet Dinner: 7:30 PM

SATURDAY, AUGUST 13
Seminars: 9:00 AM to 5:00 PM
Exhibit Hall Open: 9:00 AM to 5:00 PM
User Group Meeting: 6:30 PM
Manufacturers Hospitality Suites Begin: 8:00 PM

SUNDAY, AUGUST 14
Brunch: 11:00 AM
Seminars: 1:00 PM to 5:00 PM
Exhibit Hall Open: 1:00 PM to 6:00 PM

MONDAY, AUGUST 15
Open Discussion (optional) 9:00 AM to Noon
Microware Open House 1:00 PM to 5:00 PM

PLACE:

Des Moines Marriott Hotel. 700 Grand Ave., Des Moines, IA 50309
Telephone 515-245-5500

REGISTRATION AND REGISTRATION FEES:

You MUST pre-register! The cost is $100 per person, which includes: Reception and Buffet Dinner on Friday evening, tuition for all sessions, and the Brunch on Sunday. Please fill out the enclosed registration and return it with your check no later than July 15, 1983. Sorry, no Net 30 company purchase orders can be accepted.

ACCOMMODATIONS:

Microware has arranged a special room rate with the Marriott Hotel of either $62 single or $72 double occupancy per night. To obtain this rate, please return the enclosed card directly to the hotel as early as possible. If you are outside the U.S. you can phone or telex the nearest Marriott International Reservation Centers to make your reservations. Be sure to specify that your reservation is for the Microware User Seminar in order to obtain the special rates.

Marriott International Reservation Centers

LONDON: (01) 836-8521 Telex 24678
FRANKFURT: (0611) 23-30-41 Telex 416829
TOKYO: (03) 2157285 Telex J23376

In the event the hotel is fully booked by the time you make your reservation, Microware can assist you with reservations at other nearby hotels.

TRAVEL:

Des Moines is served by the following airlines: United, TWA, Frontier, Ozark, Republic, NWA, and Air Midwest, with direct flights from Chicago, Denver, St. Louis, Kansas City, New York and Los Angeles. Remember that many airlines offer substantially reduced fares if reservations are made at least a month in advance.

IF YOU HAVE QUESTIONS:

If we can be of assistance in any way please contact Jeanne Kaplan at Microware, telephone 515-279-8844 or telex 910-520-2535.

OS9 MAXIMUM EFFICIENCY
or
Getting More Bang For Your Buck

Paul Burega
1 Pleasant Bay
Winnipeg, Manitoba
Canada R2K 0C9
204-339-3309

OS9 is an excellent operating system. It has properties which rival large systems in ease of use and performance; e.g. I/O redirection, which allows changing the input or output of a terminal to another terminal, printer, or disk, and multiple tasking, enabling several programs to run concurrently (including several terminals with people on each of them, giving time-sharing).

Like the big systems, OS9 can be "tuned" to provide maximum performance. This article describes several ways of customizing OS9 Level I and Level II to your particular environment. By customizing, I am referring to including new device descriptors for terminals, modems, printers, and disk drives. For example, do you know that your disk drives probably can seek faster than OS9 is currently running them. Would you like to change the attributes of the ACIA controlling your terminal or printer, change the parity, stop bits, or divide ratio; or even add another disk drive or terminal descriptor. These and other items will be discussed and examples given so that you may modify these. We will also look at getting the maximum use of memory modules, one of the best ideas Microware could have put into OS9.

OS9 was specifically designed for the small system. Small systems generally have limited amounts of memory (56K for Level I, 1024K for Level II). To cope with the small amount of memory, Microware designed OS9 to make maximum use of memory. OS9 requires all executing programs to reside in memory during execution. It does not perform "swapping" of memory to disk or "demand paging" whereby the operating system moves parts of your program between main memory and disk as needed. OS9 will however dynamically allocate memory to programs which are called (thru the shell, or thru BASIC09 "run" command). For example, when you say LIST FILE, OS9 automatically loads the program named LIST into memory and starts to execute it. LIST will stay in memory until it is finished executing, and will then be deleted from memory.

However, for commands which are frequently accessed, many times the searching of the disk directory and loading of the file take longer than running the program. It would be much more efficient to load the module into memory where it will be instantly available. This reduces disk accesses and the time needed for a trivial command. Consider the command which prints out the date and time. It takes longer to load this program than it does to print the date and time.

It is possible to "LOAD DATE" into memory, and repeatedly call it without the need for any disk access. This can be done with many of the frequently used commands such as DEL, RENAME, LIST, LINK, LOAD, DATE, MAKDIR, FREE, MFREE, MDIR, DIR, etc.

There are two problems with this. As this list of frequently used progr s grows, it takes longer to load them in manually. One can put the LOADs in the STARTUP file, so they will be executed upon booting. A word of caution: the first file loaded must be the LOAD command. If it is not, then for each LOAD command, the directory must be searched for the LOAD command, then searched for the actual command to be loaded. Thus loading 10 files would require searching the CMDS directory 20 times!. If LOAD is loaded first, the number of searches is then reduced to 10.

The second problem with this approach is that there is a minimum amount of memory which must be allocated to each program. In OS9 Level I, that amount is 256 bytes, while in Level II, it is 2K or 4K bytes. Clearly, loading in a 35 byte routine will waste 230 bytes in Level I, and essentially 2K or 4K in Level II. For large programs this is not a problem. Consider Basic09, which is over 20K long. It does not matter that the last 200 bytes are not being used, because 20K is. But, for the small routines in the CMDS directory, the amount of space wasted is in most cases greater than the amount of space used. To give you a concrete example, try loading in 8 routines, such as LIST, LOAD, MERGE, DEL, LINK,

COPY, SLEEP, ECHO. You will see that 2K of memory is being used up in Level I, 16K or 32K is being used up in Level II. Quite clearly, these 8 commands really only need a few hundred bytes!!!

Other articles have mentioned burning an EPROM with several utilities in it. I originally did this, managing to include 16 commands into a 2K rom. But required several days typing in the hex codes for these commands, and checking them. I have no EPROM programmer software which runs under OS9. This does work quite well. It is now possible to boot quickly, as OS9 realizes that these routines are in ROM. As a matter of fact, since they are now in ROM, there is no need for them on the disk. Accordingly, I deleted them from my boot disk, BUT NOT FROM THE ORIGINAL OS9 DISK, OR THE BACKUP OF IT. Remember, always have a backup of your boot disk before you attempt to do something as drastic as delete commands or modify your startup or boot file.

While the EPROM route works, it is still not optimal because:

1 - it takes a long time to burn the EPROM
2 - if Microware updates any commands, you must then burn a new EPROM.

What then is the best way? What is really required is for all these small commands to be loaded when you boot, not from the STARTUP file, as this takes too long, but from the boot file, OS9BOOT. This may not seem too easy, because if you load the necessary files, then do a COBBLER (Level II, none of these commands are saved. The answer, however, is found in the documentation to the OS9GEN utility. The function of OS9GEN is to "make a copy of an existing boot file, to add modules to an existing boot file, or to create an entirely new boot file". Now, users of Level I will say, but it states "On OS9 Level One systems, the 'cobbler' command is usually a convenient way to make an exact copy of the existing boot file". Note that it states "exact copy". What we wish to do is "add modules to an existing boot file". I am not surprised that Microware did not document what I am about to present. It took me a long time to see this, but once I did, I was amazed at the results. Here, then, is a step-by-step approach to create a new, Level I boot disk, with many utilities loaded at boot time, with each utility taking up exactly the required amount of memory and no more. Level II users, be patient as some of this is relevant, and the other requirements are explained later in the article.

Required items needed before beginning this undertaking are:

1 - a freshly formatted OS9 disk
2 - a DIR X E listing, to give all the commands and their length.
3 - about 10-20 minutes of your time.

Place the freshly formatted disk in /D1 and your normal boot disk in /D0. Next, on the DIR listing, check the commands you will like permanently in memory. (I settled for all commands under 200 hex long, or which were frequently used such as FREE, TMODE, LOGON). With that list in hand, enter the following sequence of commands, [list 1], for illustration purposes, I will include the commands LOAD, LIST, LINK, LOGIN, TMODE, DATE, TSMON. You are free to use which ever you wish.

```
OS9: os9gen /d1
/d0/os9boot
/d0/cmds/load
/d0/cmds/list
/d0/cmds/link
/d0/cmds/login
/d0/cmds/tmode
/d0/cmds/date
/d0/cmds/tsmon
(ESC key, or whatever EOF is set to)
OS9:

   [list 1]
```

Once the OS9 prompt returns, the disk in drive 1 (/d1) is ready for booting. It needs the CMOS directory, SYS directory, a STARTUP file, and perhaps a DEFS directory. Any of the commands included during the OS9GEN are not needed on this new disk. I personally use the DCOPYS procedure given in the July, 1982 issue of 68' Micro Journal (p.13). This also requires a routine labeled FIXS, which was in the May, 1982 issue of 68' Micro Journal (p.16). I have modified it to run quickly by adding a #40 to the copy commands passed to the shell. When in BASIC09, with DCOPYS and FIXS loaded, do a $MFREE, and see how many pages are left free. The more that are contiguous, the faster the copy will proceed. I chose 40 pages as this used up almost all the rest of my available memory. You can release some of BASIC09's extra memory by using the MEM command. You could say MEM 2000 or less, and that will return some of the memory to OS9. You will note that I also do a LOAD on the COPY command, so that it stays resident for the entire use of the directory copy, then UNLINK it at the end. I personally chose the following commands to include at boot time: ECHO, DEL, LINK, MAKDIR, LOAD, UNLINK, SLEEP, LIST, TSMON, COPY, DISPLAY, DATE, BUILD, MERGE, PRINTERR, SAVE, DIR, MDIR, MFREE, PROCS, FREE, RENAME, TMODE, VERIFY, DUMP, LOGIN.

Note that I no longer have all those commands on my current boot disk. Now with few commands left in the CMOS directory, I have free space, a much needed resource on my single-sided, double-density disks. However, I still have my original OS9 disk, and a second copy of my old boot disk, both of which have these commands on them. Note that the only way to delete one of the commands from the boot file, is to remake the boot file from the original OS9BOOT. However, new versions of files may be included at the beginning, before the current OS9BOOT, then a new OS9GEN, but this will waste some space. For small commands, or for new device descriptors, this does not make much difference. With this new boot disk, the boot process should be very fast and your startup file may shrink in size. My new startup consists of:

```
SETIME 83
DATE T
ECHO OS9 Level I successfully booted.
```

Once you have made all the directories, and copied the necessary files, you are ready to try this new boot disk. Put the disk in /d0 and hit reset. OS9 should come back quickly, with much less disk activity needed for initialization. Now type MDIR. Note how quickly you get a response (assuming you included it in the commands in the boot). Note how many commands are sitting there to give you instant response. No need to wait for disk delays. Now thats speed!!! This is where OS9 really shines. The other 6809 operating systems do not really provide this ability to have all these modules present (at least not in a relocatable sense). Now do an MFREE, the amount of free memory will be less than you are used to, but you have all these extra commands sitting there, which now you will (hopefully) use more often. If you had previously LOADed these files, you will notice a net gain in memory. It certainly is nice that OS9 contains these options. I found that these commands require about 8k of memory. Since I used to load most of these with LOAD, there is not that much more memory being used. I feel that the increase in speed outweighs the memory lost. With memory getting cheaper, it is better to utilize this fully, and get that lightning speed from the operating system. "Sure beats the pants off all those other systems"!!

Some Notes on this Level I conversion:

The new disk is required because the boot program must be contiguous on the disk. It cannot be scattered across the free space of the disk like a normal file. Using a new disk also ensures that you ke the old disk as a security backup. Also, using a new disk ensures the the file OS9BOOT is the first file on the disk, thus the disk does not have to seek very far to find the boot.

If you wish to change the boot program, you can make a new boot on this new disk, as all the remaining space will be contiguous after the files are copied. You can check this by using the FREE command. If the largest block reported is the same size as the amount of free space, then there is only one piece of free space. Don't worry if it is not, as long as it is within a few sectors. If it is not contiguous, then OS9GEN will give an error message saying that you will not be able to boot from this disk. DO NOT Despair. Rename OS9BOOT to xxx and recall OS9GEN, providing there still is enough free space left on the disk. If so, then this OS9GEN may work, and then delete the xxx file. You may have to rename the file TEMPBOOT to a dummy name as well. If you still have problems, then you will be best off formatting a new disk, and trying again. But, if you just want to add more commands to the boot, you can use the modified OS9BOOT file, which has some extra commands already in it, and do not re-include those files again. Or, you can go back to the original OS9BOOT file, and re-specify all the files wanted.

Notes for OS9 Level II

Level II users should not encounter all the troubles of Level I users have. Level II supports loading files which contain a number of routines, and it will load them

contiguously. But, again we find the fragmentation problem. If your version of Level II uses 2K or 4K blocks, then make sure the files you load are each just under 2K or 4K in size. If they are just over 2K or 4K (depending on your particular page size), then you will be wasting memory. Smoke Signal systems use 4K page sizes, and I found that it is possible to group a set of commands so that they are just slightly under 4K in length (EXX or FXX in length). It is better to make a number of smaller files than to load one large one. If a program requests one of the programs in a block, its 64K address space will lose the amount of memory equal to the size of the block(s) of memory containing that program. For example, if you decided to merge a number of commands into a 9K file, OS9 would use 12K to store it once loaded. Now, when a program links to one of those programs, it would lose the ability to refer to 12K of its 64K memory, since 12K of the memory it could see would be taken up so that it could "see" that program. However, if that program was in a block, just under 4K long, then only 4K out of the 64K would be used up. Note that if you just load a 50 byte routine, it will also take up 4K, so it is best to merge some utilities. It is best to merge utilities which may be used together, so that if both are needed concurrently, then only 4K will need to be used.

How do you create these files to load. Through the use of the MERGE command. Just enter:

```
OS9: chd /d0/cmds

OS9: merge list link tsmon login makdir rename del dir
mdir >util

OS9: merge rename sleep build copy attr procs free
mfree >util2
          etc.
```

Check the length of the files to ensure that they do not go over OFFF using the E attribute of the DIR command. Note that merge does not set the execute bits of the output file (in this case UTIL1, UTIL2). To enable OS9 to load these files, you must issue the command:

```
OS9: attr util1 e pe
```

Note that this only need be done once, and then will show up on the DIR E command.

Next, you must modify your startup file to do LOADs on these files. The first few lines in your new startup file should be:

```
LOAD UTIL1
LOAD UTIL2
```

followed by the rest of your normal startup, except that you can now delete the other loads which refer to programs already in the UTIL files.

Which files should be included in the UTIL files. I included the following:

```
UTIL1:
   attr, build, copy, date, del, dir, display, dump, echo,
free
   link, list, makdir, merge, rename  ** total length F97

UTIL2:
   login, mdir, mfree, procs, setime, sleep, tee, tmode,
tsmon,
   unlink ** total length FA2

UTIL3:
   save, verify, version  ** total length 46E
```

However, two important files have been omitted. These are the commands LOAD and SHELL. It seemed a pity to use up 4K just for a 50 byte LOAD, while SHELL is around 1K. I decided to put both in the OS9BOOT file using the above procedure for OS9GEN. I found that this freed up 12K on the system. Now while memory is getting cheaper, most Level II users do not have 1024K installed in their machine, perhaps 128K or 192K. The extra 12K or 16K still makes a difference (Basic09 will almost fit in the new free space!).

Once you follow the rest of the instructions for copying over to a new disk the remaining commands under the CMDS directory, as well as SYS and DEFS, and make the new startup file such as:

```
load util1
load util2
load util3
setime 83
date t
tsmon /t1 &
echo OS9 Level II system initialized
```

Then, go and boot with this new disk. You will be amazed at the faster boot which does less disk seeking. You will also be able to issue most commands in the OS9 manual without reading them in from disk. This will really impress people who are used to large systems, especially if you let them LOGIN on another terminal, and you simultaneously execute programs. Try doing a MDIR E. You will find that the commands will now be nicely clustered in 4K pages, and will be contiguous in those 4K blocks. Enter MFREE. See how much more of that valuable memory you have available.

By spending time customizing your copy of OS9, you are able to greatly increase the speed of execution of various functions. You are also able to reclaim varying amounts of memory, and get more use out of your system. I had spent many we s banging my head against the wall trying to accomplish what I finally did. I feel that there must be others who are not familiar with what can be done to improve OS9. You may figure that it may not apply to you, or that it may not be worth the effort, but I assure you that it is worth the effort to get the system to work quicker and more smoothly.

If you have ever worked with larger systems, with fast hard disks, you probably get annoyed sometimes at your slow 5 inch floppies. The method presented adds that extra oomph to your system, and you may decide that it appears as if you have DMA hard disks on your system.

Modifying Device Descriptors in OS9

If you are like me, you find that your version of OS9 does not run the disk stepper motor at the fastest rate that it can handle. This is especially true if you have several drives, each of them different in this respect. Your drive stepping speed is probably set to the slowest so that all your drives will work, or perhaps the initializer byte for the ACIA on term or p1 is not what you would like it to be. You may wish to change the address of P, P1, T1, or even create another port, or disk drive. This article will cover all this using several approaches, for both Level I and Level II.

When I first obtained OS9 for my system, I was very frustrated because the disk stepper motors were running at the slowest speed. I have two MPI drives which can run at 6ms step speed. OS9 was running these at 30ms step speed. Operations, such as searching the disk directory, do a large amount of disk seeks. By speeding up seek speed, the speed of these searches is enhanced. In Smoke Signal DOS, this function is accomplished through the use of the LUNCTL command which allows setting the seek speed as well as the type of drive (5" vs. 8", 40 vs. 80 track, etc.). Unfortunately, no such command exists under OS9. No mention is given on how to change these functions other than the fact that the device descriptor module contains a value for this (p. 6-8).

I had purchased DEBUG, so I decided this was as good a time as any to try it out. I found that I could modify values in the device descriptor, and the drives starting seeking faster on seek operations. Then I thought, "What happens next time I boot". I COBBLERed a disk, so that it would contain the new stepping rates. When I attempted to boot the disk, it started the boot, then died. It took me some time to figure out that because I had modified the device descriptor, its CRC was no longer correct. That meant that OS9 would not use the module after a boot. Good thing I had a backup disk to boot from!.

The solution to this problem came after a tremendous amount of thought. The VERIFY command will update the CRC for a module. If you save the modified module to disk, update it to another file, then load that file; this method will not work, because if you attempt to load a file which already exists, it can only be replaced with a higher revision number. Go back into DEBUG, and bump up the revision number of D0 to 2 from 1. Then save, verify, use attr to set the execute bits, and you can now load these new disk descriptors. Each descriptor would be loaded on a 256 byte boundary! (where have we discussed this problem before)? I had not yet figured out the OS9GEN approach, so for a time I had a number of LOAD statements in my startup file for the modified drives. I decided that if DEBUG can update bytes in a module, then it could also write in the correct module headers and trailers. This was getting easy! All I had to do was DUMP the verified modules to the printer, then use debug to change all the values in the module. Now I could COBBLER a new boot disk, and this disk would boot successfully.

However, this approach is not suitable with Level II, and will not readily create new device descriptors. What we really need is to create a copy of the descriptor we wish to modify on disk, then perform changes to that copy, and finally perform an OS9GEN to create a new bootstrap file to incorporate these changes. The first part is easy. As an example, let us modify T1. We will change the port address, as well as the default initialization to include 2 stop bits instead of one, with 8 data bits.

First step is to make a copy on disk. This is done via the SAVE command.

```
OS9: save t1
```

This will create a copy of t1 in the current data directory.

Next we require a hardcopy dump of the descriptor, to ascertain what we will modify.

```
OS9: dump t1 >/p
```

This creates a nicely formatted dump on the printer.

T1 is an SCF device. Page 7-6 of the OS9 System Programmers Manual describes the locations of various function bytes. The port location is not listed. This is at offset $0E, which is a 3 byte address (for extended addressing). Typically it will be the third byte which you wish to change, at offset $10. The ACIA initialization value is at offset $26. To get 8 data bits, 2 stop bits, divide by 64, the value is $12 (taken from a Motorola data sheet). Microware states that it's a good idea to increase the revision level, as OS9 keeps the module with the highest number. Most modules are at revision level 1 (bottom four bits in byte 7). Leave the top four bits alone !!!

Now, we run program (given later on, entitled DISKCHNG) which will allow us to modify bytes in file T1, which we created. This program allows changing only 1 byte at a time, so you must re-enter the file name each time if you have more than 1 change to make. Basic09 allows entering hex data by preceding the data with a "$". Thus, the sequence of commands entered is: (note: do not enter the comments on the right starting with rem, they are just to explain what is going o in each section!)

```
OS9: basic09
B: load diskchng
B: run
FILE NAME t1
ENTER BYTE TO CHANGE $07          rem: change
revision level
THE VALUE IS  81
ENTER NEW VALUE $82
THE VALUE IS 82
FILE NAME t1
ENTER BYTE TO CHANGE $10          rem: change
port address
THE VALUE IS  E0
ENTER NEW VALUE $e2
NEW VALUE IS  E2
FILE NAME t1
ENTER BYTE TO CHANGE $26          rem: change
initialization
THE VALUE IS 15
ENTER NEW VALUE $12
THE VALUE IS $12
FILE NAME  (control Q to quit)
B: bye
OS9:
```

File T1 has had changes made, and no longer is a valid OS9 module, and must be verified.

```
OS9: verify u <t1 >new.t1
OS9: attr new.t1 e pe
```

This command line tells the verify command to do an update (u), take the input from t1, and write a new file called new.t1. new.t1 will then contain a valid OS9 module. It can now be OS9GENed into a new boot file, or just loaded (since the revision number was updated, OS9 will keep around the copy with the highest number). The attr marks the file as executable.

Last step is to create a new boot file (follow procedure given before for including extra modules in the boot file). Note: the file name you wish to include is NEW.T1, not T1. If you include T1, then the CRC will be incorrect, and OS9 will ignore it.

Modifying a Disk Descriptor

Disk descriptors can be modified in a similar manner. For RBF descriptors, such as D0, refer to page 6-8 in the OS-9 Level One System Programmer's Manual (I have not seen a Level II manual, as versions of Level II that I have seen, come with Level I manuals). There are several values which you might wish to change for each drive. One of them is the seek speed (offset $14). A table is given at the bottom of the page giving the step codes and their corresponding step speeds. Offset $13 contains the drive number (in case you wish to make D0, drive 3 not a really good idea). This is useful though if you wish to create another drive. All that needs to be done to create another drive is to save a drive descriptor, say D0, then dump it to the printer as above. Note the locations of the character string "D0", and modify it to be "D3" or whatever two character name you wish. To extend the length of the name requires reassembly of the descriptor (If you can create the source to it in the first place !), or much headaches if you try and modify the name yourself without assembling the table. To create new terminals, modify T1 by changing the name "T1" and the po t address. Simple once you know how. Note well; character strings in OS9 have the high bit on in the last character in the string, so that a "1" is $B1, not $31. Here then is an example creating a new drive from D0, and increasing the step rate.

```
OS9: save d0
OS9: dump d0 >/p
OS9: basic09
B: load diskchng
B: run
FILE NAME d0
ENTER LOCATION $13                rem: change drive
number
VALUE IS  0
ENTER NEW VALUE 4                 rem: from 0 to 4
VALUE IS  4
FILE NAME d0
ENTER LOCATION $14                rem: change step
speed
VALUE IS  0
ENTER NEW VALUE  3                rem: from 30 ms to
6 ms
VALUE IS  3
FILE NAME d0
ENTER LOCATION $22                   rem: this value
may change
VALUE IS  80                      rem:  econd byte of
"D0"
ENTER NEW VALUE  $b4                 rem: change to
"4"
VALUE IS  B4                      rem: giving "D4"
FILE NAME (control Q)
B: bye
OS9:
```

Again, verify must be done to create a valid module. If you only wish to change the stepping speed, then just change the appropriate value (and perhaps the module revision number). Other bytes you may wish to change are $16, media density (to allow 80 track drives to read 40 track disks). Note that if you are using 80 track drives, then the double track density bit should be set. This allows the software to read 40 track disks on 80 track drives. You can also write 40 track disks, but there is no guarantee that a 40 track drive will be able to correctly read all of the disk, as the 80 track heads write a much narrower band of data. It is an advantage to be able to read 40 t ack disks on an 80 track drive. This feature is especially useful for people who have a mix of 40 and 80 track drives in the systems, or people who receive 40 track disks from others. WARNING: If you do not have this bit set and have 80 track drives, you will not be able to read those disks once the bit is turned on. When the bit is off, the disk has written on it information which says it was created in a 40 track drive, with 80 tracks!. Hence, when you tell OS9 that your drive is 80 track, it will look at the disk, see that it is 40 track, and skip every other track, and of course will give seek errors. If you have two 80-track drives, then it is possible to fix up one drive to be correct, format some new disks, and then copy all your disks. Make sure you then fix up D0 if it is 80 track, and create a new boot disk if necessa y.

Another reason for setting this bit correctly, is that if a disk written on an 80-track drive is inserted in a 40-track drive, OS9 will generate an error message (wrong type media). Another value to change for 80 track drives is the 2 byte number at $17-$18, which contains the default number of tracks for that drive, and location $19, which contains the number of heads for the drive (giving the number of surfaces). These bits are used by the format utility to pick up a default disk type. Do not modify bytes $1A - $20. Someone intelligent provided values for these, and they should not be changed, except by a very knowledgeable person.

OS9 users will become expert at customizing systems to their particular needs. I hope that some readers can use this information, and perhaps discover an easier way of making the modifications I have described.

```
PROCEDURE dcopys
 0000    REM copy dir files
 0011    REM space to copy, any other key to bypass
 003A    REM from July, 1982 '68' Micro
 0057
 0058    DIM cmd$:STRING[128]
 0064    DIM c$:STRING[1]
 0070    DIM p:BYTE
 0077    TYPE rec=name:STRING[29]; misc:STRING[3]
 0092    DIM drec:rec
 0098    INPUT "from dir > ",fd$
 00AE    INPUT "to dir > ",td$
 00BF    SHELL "load copy"
 00CC    CHD td$
 0001    PRINT "space to copy - any other key to bypass"
 00FC    PRINT
 00FE
 00FF    ON ERROR GOTO 10
 0105
 0106    OPEN #p,fd$:READ+DIR
 0112    WHILE NOT(EOF(#p)) DO
 0115      REM read next dir entry
 0136      GET #p,drec
 0140      IF LEFT$(drec.name,1)>" " THEN
 0153        IF LEFT$(drec.name,1)<>"." THEN
 0166        RUN fixs(drec.name)
 0173            cmd$="copy "+fd$+"/"+drec.name+"
"+drec.name+" #40"
 01A0        PRINT cmd$; " ";
 01AA        GET #0,c$
 01B3        PRINT
 01B5        IF c$=" " THEN
 01C2          SHELL cmd$
 01C7        ENDIF
 01C9        ENDIF
 01C8      ENDIF
 01CD    ENDWHILE
 01D1    CLOSE #p
 01D7    SHELL "unlink copy"
 01E6    BYE
 01E8 10   REM error routine - bypass and try again
 0212    PRINT "error = "; ERR
 0220    GOTO 5
```

```
PROCEDURE fixs
 0000  REM : this routine fixes up a string
 0023  REM : a string read in from the directory has the
8th bit
 004F  REM : turned on in the last character to signify
end of string
 0098  REM : basic09 signifies the end of a string with
hex FF
 00CE  REM : this routine fixes up the string to end in
255 decimal
 0112  PARAM x(29):BYTE
 011E  DIM I:INTEGER
 0125  I=0
 012C  LOOP
 012E    I=I+1
 0139  EXITIF x(I)>128 THEN
 0148    x(I)=x(I)-128
 015A    x(I+1)=255
 0168  ENDEXIT
 016C  EXITIF I>27 THEN ENDEXIT
 017B  ENDLOOP
 017F  END
PROCEDURE diskchng
 0000     REM program to change by es in a disk file
 0029     REM by Paul Burega
 003A     DIM path:BYTE; name:STRING[50]; number:BYTE
 0052     LOOP
 0054       INPUT "file name ",name
 0066       OPEN #path,name:UPDATE
 0072       INPUT "enter byte to change ",ptr
 008F       SEEK #path,ptr
 0099       GET #path,number
 00A3       PRINT USING "s20,h2"," he value is",number
 00C1       SEEK #path,ptr
 00CB       INPUT "enter new value ",number
 00E3       PUT #path,number
 00ED       PRINT USING "s20,h2","new value is",number
 0108       CLOSE #path
 0111     ENDLOOP
```

STRUCTURED
ASSEMBLE MACROS

Yves Leclerc
320 Landreville #1A
Verdun HJE 1B4
Quebec, Canada.

STRUCTURED ASSEMBLER THROUGH MACROS

I'm sure a number of You read with some interest, a year ago, Greg Walker's articles in BYTE about a structured assembler for the 6809. And thought, as I did: Wouldn't that be nice... if I understood how it works, or if it was written for the TSC rather than the Motorola assembler!

The advantages would be obvious: cleaner and clearer structures in medium-size and long programs; easier writing, easier debugging; better modularity. My problem was that I had never really worked with macros or conditional assembly, and just looking at the Walker listings made me very much aware of how hard a job it might be.

Walker's method is one that is used sometimes on larger systems, but rather rare on micros: coding a language, or in this case language extensions, in assembler macros. It is similar to the technique used by H.L.Harkness to create Rload ('68 Micro sept-nov 1981).

It took me some six months to make up my mind, then sometime last summer, as I planned to embark on some fairly fancy assembly programming projects, I figured I might save more time tackling this task first and thus giving myself a powerful tool to help solve the other problems. It did take quite a bit of work (so much that I'm not sure I saved time overall), but it was extremely instructive... and now I wonder how I could do without structured assembly for most of what I write.

For those who haven't seen the Walker articles, I'll give here a quick run-down on macros and conditionals, with both their advantages and inconvenients. Experienced assembler hackers and those who have read Walker may skip the next two sections.

MACROS

Macros are blocks of instructions given a name, somewhat like subroutines, except that each time they are called (just by writing their name in the program), they are "expanded" or written again into the text. They're good either for short series of instructions used often in a program, or for blocks that are not quite identical, but almost, from one use to the next.

For instance, if you write an assembler program doing quite a bit of file juggling, you might write a "family" of macros called OPEN, READ, WRITE and CLOSE to take care of the details of file access. Or if you use multiple precision

arithmetic in a program, you could have PLUS, MINUS, TIMES and DIVIDE macros taking care of that.

Macros are usually defined at the beginning of the program, and after that they can be considered simply as new, more powerful or more complex, instructions of the assembler. Another good method (the one I use here) is to have a library of macro files you can call through the LIB pseudo-instruction of your assembler. That way you won't have to rewrite the same series of instructions each time you need them.

On the other hand, the first times you use macros, it is hard to remember that they are not part of the program, and are not executed unless they are specifically called. They are used at assembly time, not at run time. Which at first can lead to a number of hard-to-track errors.

Macros can also have parameters, or arguments. Except that these parameters are litterals, not values or expressions. A macro parameter, in TSC format, is referred to in the body of the macro by an ampersand followed by a number from 1 to 9 indicating the order of the parameter in the calling line. For instance, if a macro PLUS is called as

```
PLUS VALUE,+,D
```

the expression &1 will refer to the ASCII string "VALUE", &2 will become "+" and &3 will be "A" (which could as well represent the letter "A" as the hex value 10 or the A register of the 6809!).

Another difficulty is that you can't use labelled lines inside a TSC macro: since each label will be "created" anew each time the macro is expanded on pass one of the assembler, it will generate a "multiply defined symbol" error. The only (not very useful) way to go around this is to give the macro new label names each time as parameters.

CONDITIONAL ASSEMBLY

I had read about this in the TSC assembler manual and in Leventhal's book on the 6809, but I must admit I had not the least idea what it could be used for, except in a very limited way. That's because I didn't understand the difference between what happens to a program at assembly time and what happens to it at run time.

So here we go: conditional assembly has no effect at all on what happens when you run a program. Its influence ends as soon as the program is assembled into binary form. What it does is assemble the program, or a program section, differently in different circumstances according to a set of rules. One of its major uses is inside macros: a macro can be defined in various ways according to the parameters, or even to the number of parameters, passed on to it. This is the way I use conditional assembly in the present series of structured macros.

But since conditional assembly is restricted to assembly time, values created by a run of the program cannot be used to set the conditions; this is specially true of variables or of the contents of memory bytes, since there is no way you can know these in advance at assembly time. Thus if your conditional assembly depends on certain numerical limits that vary from one invocation to another, the only way to pass them on is through explicit expressions or pseudo-variables created by the assembler at assembly time. But since you can use EQU on a symbol only once in a program, you have to use the SET pseudo-directive instead, which can be redefined at will.

In practice, conditional assembly is a very powerful technique, but it also has severe drawbacks: it can make the code impossible to read and understand, and introduce any number of weird bugs, often caused by a confusion between assembly time and run time behaviour. So I would tend to side with Lance Leventhal, whose advice about conditionals is "Don't use... unless you have to".

LEVELS OF MACROS

Let's now go into the Astruc family of macros itself. It is built in four levels: primitives, tests and code generation, stack control, and the structured macros themselves. Each level can and does call the preceding levels to execute part of its tasks. But as these will vary from one call to the next, I have to use conditional assembly a lot, which tends to obscure the listings and thus the role of each macro.

This doesn't matter if you want to simply transcribe the macros and use them as such: the programmer only uses the last level in his programs, and does not need to know how they work. But if you want to change them, or add on to them, you had better know what you're getting into. So I shall describe briefly what each of the first three levels does.

a) Primitives: they do strictly low level short and simple tasks used by the higher level macros. LGT2T sets pseudo-variable BR1LG to 0 or 1 according to whether the L parameter for long branching instructions is present. The others are all concerned with stack management, about which more in a moment.

b) Tests and compares: they test and validate the various parameters of higher level macros, and generate assembly code accordingly. They check for instance that the registers and conditions referred to in the fourth level macros are correct, and "write" into the program the needed conditional branching instructions, often leaving the adress field blank or filled with a dummy value, to be defined later in the process (the RP1LG pseudo-variable is used only for this).

c) Stack control: to be able to nest assembly structures inside each other we need a stack to check at which level we are and to hold values and addresses. But the conditional assembly format does not allow this, so we must simulate a stack using dummy variables.

This is probably the hardest part of the job of creating the structured macro family. First, you need simple and easy to use PUSH and PULL instructions, to be called at the highest macro level, and to take care automatically of stack management tasks. These have only two possible parameters: an address, and the Q option indicating a more complex PUSH or PULL required by the multiple choice structure CASE OF.

Then (and this is a major difference from the Walker structured macros in BYTE) we have another macro, at the primitive level, EX9MAC, which is called every time by the P&SH and P&LL macros to determine at which level of the stack we are. This has to be done "by hand", testing each possibility through a binary search, which makes EX9MAC probably the most complex of all the macros here.

Finally, once the stack level is determined, the primitive P2SH or P2LL utility macros are called to effect the stack manipulations themselves. There are two versions of each, the simple one, which acts on only one variable, and the "Q" one, which acts on three.

STRUCTURED ASSEMBLY

The fourth level of macros is the one which transforms your standard TSC assembler into a quasi-higher level language, without robbing you of any of its versatility and hardly any of its compactness. Instead of a structured macro, you can always use a "normal" compare and jump in any situation, but based on half a dozen programs, I figure that the extra code generated by the macros is never more than 5%... and the extra running time is about nil: most of the unnecessary instructions never get executed anyway.

Structured assembly has three main advantages: it gets you rid of all the confusing jumps and branches (conditional or not), thus making the program much more understandable and easier to debug -- in fact, I find that most of the time I save is in the debugging rather than the coding stage -- and test; it eliminates the need for most line labels, since almost all the jumps are generated directly by the macros; and it regroups the program parts into functional blocks in a much more logical way, so that program flow "jumps around" much less and is more obvious.

As in any high-level programming language, you use the structured macros in two circumstances: when there is a choice to be made between two or more alternatives, and when you need a loop repeated from zero to a number of times or until a certain condition is met.

CHOICE STRUCTURES

There are two families of structured macros used for choosing. One, the IF family, serves for choices between two alternatives; the other, the CASE family, is used when there are more than two possibilities.

The IF family holds five macros: three different forms of IF, the IFNOT (meaning ELSE) clause, and the IFEND macro indicating the limit of the scope of an IF structure. Why three IF forms? To take advantage of the variety of testing instructions on the 6809, and to optimize the resulting code. The IFRG form compares between an internal register and a value in memory, using equal/not-equal, greater-or-equal/smaller, and greater/smaller-or-equal tests. The IFTST form checks, using the TST instruction, a value held in memory or the A or B register, and branches accordingly. The IFCC form uses the status of the condition code register, which should already have been set, to make a choice.

Each form of the IF macro can be followed by a block of code to be executed if the test is successful, an IFNOT clause, and another block of code to be executed if the test is not successful (the IFNOT part is optional). The end of the whole structure (which can be nested) is marked by a IFEND macro.

The CASE family has four macros, and allows for one of many blocks of code to be executed according to the result of an equal/not-equal test between an internal register and a value in memory. It also has an OTHER default case to take care of situations in which none of the tests succeed. Its end is marked by the ENDCASE macro.

Note that the IF macros have the same form as in the Greg Walker BYTE articles, but they are coded differently: they nest 16 deep instead of 10, and the stack management is done not inside them but by a lower level group of macros. The CASE macros are new... and pretty useful especially where for instance a value or a character in the A register has to be tested for several possibilities. You will see typical uses of these forms in the listings.

LOOP STRUCTURES

There are three loop structure macros, two adapted from Walker (WHILE... ENDWHILE and REPEAT... UNTIL), and a new one, a counting loop similar to the BASIC or Pascal FOR NEXT loops. The REPEAT loop is the simplest form: it generates a test-and-branch at the end of the block, without any overhead. The WHILE form is more complex, since it tests at the beginning of the block, and thus generates a conditional branch at the beginning and an absolute branch at the end.

The FOR loop is the most complex, and often the most useful: it uses one of the internal 6809 registers as a counter, which is incremented or decremented at each execution of the enclosed block of code, until it reaches a preset limit. If the increment (or decrement) is 1, the INC instruction is generated if possible, else the ADD or LEA instruction is used. For practical reasons, the initial value and limit are set at the beginning of the loop (where the test is made), but the increment is defined at the end, as a parameter to the NEXT macro.

All these structures can be nested inside each other up to 16 deep, the pseudo-stack taking care of the details,

thus allowing the programmer to concentrate on the meaning rather than the mechanics of the whole program and its various sections. "Pretty-writing" the code with indentations in Pascal or C fashion makes the architecture of the program much clearer, and the flow of control much easier to follow to the author himself.

LISTINGS AND EXAMPLE

You will find appended to this first the listings of the structured macros themselves, then a program example of their use in practice. To use the structured macros, you copy them in a text file called ASTRUC.TXT, and when you write a program, put LIB ASTRUC at the beginning. You may then use any of the macros as you would other "natural" instructions of the assembler.

Note that all 6809 addressing modes may be used as parameters to these macros (wherever an address is needed), and that the responsibility to force long branching through the L parameter rests with the programmer, although the macros will send out an error message if long branches are needed but not present. More detailed information on the syntax is included in the listing comments before each macro.

The rather bizarre labels chosen for the lower level macros and the pseudo-variables have no real meaning. I made them unusual enough so there would be little risk of conflict between them and any other labels the user could create in a structured assembly program. Thus P6LL and P6SH instead of PUSH and PULL, etc.

The second listing is a short example of a structured assembly program, the SEARCH utility to look for a series of hex bytes in any FLEX file, text or binary. Explanations of this program will be found in the comments lines and fields of the listing. The last listing is a disassembly of the program (using DYNAMITE) to show the code generated by the Astruc macros. If you compare the structured source and the results, I think you'll have to admit first that the structured form is much easier to read and follow... and second that if you had coded the same program in "straight" assembler, it wouldn't have been shorter by more than a dozen bytes or so.

If you assemble the SEARCH program, you will notice that assembly takes quite a bit more time than you expect. That is the penalty you pay for using macros in a program. But it is usually more than adequately compensated for by the time you save coding and debugging the programs. Since the macros generate a lot of ORG instructions that will be reflected in the structure of the resulting binary file on the disk, I suggest one extra step to obtain simpler code: once your program is assembled, note its limits in memory, then dump it to memory using GET and save it using SAVE or SAVE.LOW. This will get you rid of all those ORGs and result in a file that loads much faster.

-------!-------

```
          * STRUCTURED ASSEMBLY MACROS
          * Yves Leclerc, September 1982
          * Adapted and expanded from Greg Walker
          * of Motorola (BYTE, nov/dec 1981)

          * INTERNAL DUMMY VARIABLES

0000  ACP1L   SET     0
F000  RP1LG   SET     $F000
0000  BR1LG   SET     0
0000  RG2DD   SET     0

          * PRIMITIVES
          * ==========

          * TEST FOR LONG BRANCH

LGT2T     MACRO   [L]
          IFC     L,&1
BR1LG     SET     1
          ELSE
BR1LG     SET     0
          ENDIF
          ENDM

          * PUSH ON STACK
          * a) Short case

P2SH      MACRO   <lvl>,<addr>
SP1L&1    SET     &2
LP1L&1    SET     BR1LG
          ENDM

          * b) Double case

P2SHQ     MACRO   <lvl>,<addr>
SP1L&1    SET     &2
LP1L&1    SET     BR1LG
RP1L&1    SET     RG2DD
AP1L&1    SET     AD2PL
          ENDM

          * PULL FROM STACK
          * a) Short case

P2LL      MACRO   <lvl>
          ORG     SP1L&1
BRL2G     SET     LP1L&1
          ENDM

          * b) Double case

P2LLQ     MACRO   <lvl>
          ORG     SP1L&1
BRL2G     SET     LP1L&1
RG2DD     SET     RP1L&1
AD2PL     SET     AP1L&1
          ENDM

* DETERMINE STACK LEVEL
* and execute macro

EX9MAC    MACRO   <macro>,<addr>
          IF      ACP1L<9,23
          IF      ACP1L<13,11
          IF      ACP1L<15,5
          IF      ACP1L=15,2
          &1      16,&2
          IF      ACP1L,48
          &1      15,&2
          IF      ACP1L.38
          IF      ACP1L=13,2
          &1      14,&2
          IF      ACP1L.35
          &1      13,&2
          IF      ACP1L,33
          IF      ACP1L<11,5
          IF      ACP1L=11,2
          &1      12,&2
          IF      ACP1L,29
          &1      11,&2
          IF      ACP1L,27
          IF      ACP1L=9,2
          &1      10,&2
          IF      ACP1L,24
          &1      9,&2
          IF      ACP1L,22
          IF      ACP1L<5,11
          IF      ACP1L<7,5
          IF      ACP1L=7,2
          &1      8,&2
          IF      ACP1L,17
          &1      7,&2
          IF      ACP1L,15
          IF      ACP1L=5,2
          &1      6,&2
          IF      ACP1L,12
          &1      5,&2
          IF      ACP1L.10
          IF      ACP1L<3,5
          IF      ACP1L=3,2
          &1      4,&2
          IF      ACP1L,6
          &1      3,&2
          IF      ACP1L,4
          IF      ACP1L=1,2
          &1      2,&2
          IF      ACP1L,1
          &1      1,&2
          ENDM

          * TESTS AND COMPARES
          * ===================

          * TEST FOR VALIDE PARAMETER

PAT2T     MACRO   <par>
```

```
        IFNC    &1,,3
BNP4    SET     0
        ERR     * PARAMETER ERROR *
        EXITM
        IFC     &1,L,-3
BNP4    SET     1
        ENDM

* GENERATE CASE INSTRUCTIONS

G2NOD   MACRO   <reg>,<addr>
        CMP&1   &2
        IF      BR1LG
        LBNE    RP1LG
        ELSE
        BNE     *
        ENDIF
        ENDM

* FIND CASE REGISTER

G1NOD   MACRO   <addr>
        IF      RG2OD>3,9
        IF      RG2OD=3,6
        IF      RG2OD=1
        G2NOD   A,&1
        ELSE
        G2NOD   B,&1
        ENDIF
        IF      RG2OD,10
        G2NOD   D,&1
        IF      RG2OD.8
        IF      RG2OD=6,6
        IF      RG2OD=4
        G2NOD   U,&1
        ELSE
        G2NOD   X,&1
        ENDIF
        IF      RG2OD,1
        G2NOD   Y,&1
        ENDM

* GENERATE COMPARE

OPT2T   MACRO   <cond>,<cond>,<cond>
        IFC     &1,&2
BN0P    SET     1
        IFN     BR1LG
        B&3     *
        ELSE
        LB&3    RP1LG
        ENDIF
        ELSE
        IFC     &1,&3
BN0P    SET     1
        IFN     BR1LG
        B&2     *
        ELSE
        LB&2    RP1LG
        ENDIF
        ENDIF
        ENDIF
        ENDM

* FIND CORRECT TEST

OPT5T   MACRO   <cond>,[T:C]
BN0P    SET     0
        IFNC    &2,,6
        OPT2T   &1,PL,MI
        IF      BN0P,12
        OPT2T   &1,CC,CS
        IF      BN0P,10
        OPT2T   &1,VC,VS
        IF      BN0P.8
        IFC     &2,T,2
        OPT2T   &1,GT,LE
        IF      BN0P,5
        OPT2T   &1,EQ,NE
        IF      BN0P,3
        OPT2T   &1,GE,LT
        IF      BN0P,1
        ERR     * ILLEGAL CONDITION &1 *
        ENDM

* VALIDATE REGISTER

RGT5T   MACRO   <reg>,[C:M]
BNR3G   SET     0
        IFC     &1,D,4
        IFC     &1,S,3
        IFC     &1,U,2
        IFC     &1,X,1
        IFNC    &1,Y,1
BNR3G   SET     1
        IFNC    &2,C,1
BNR3G   SET     1-BNR3G
        IFNC    &2,M,2
        IFC     D,&1,7
BNR3G   SET     BNR3G+BNR3G
        IF      BNR3G,5
        IFC     &1,A,1
        IFNC    &1,B,2
BNR3G   SET     1
        IF      BNR3G,1
        ERR     * ILLEGAL REGISTER &1 *
        ENDM

* STACK CONTROL
* =============

* PUSH

P6SH    MACRO   <addr>,[Q]
ACP1L   SET     ACP1L+1
        IF      ACP1L<17,2
        ERR     * STACK OVERFLOW *
        EXITM
        IFC     &2,Q
        EX9MAC  P2SHQ,&1
        ELSE
        EX9MAC  P2SH,&1
        ENDIF
        ENDM

* PULL

P6LL    MACRO   [Q]
        IF      ACP1L,2
        ERR     * STACK EMPTY *
        EXITM
        IFC     &1,Q
        EX9MAC  P2LLQ
        ELSE
        EX9MAC  P2LL
        ENDIF
ACP1L   SET     ACP1L-1
        ENDM

* STRUCTURED MACRO-INSTRUCTIONS
* =============================

* CONDITIONAL STRUCTURE

* There are three possible conditional
* instructions, according to the type
* of test made. But their syntax is similar,
* their operation identical: if the result
* of the test is TRUE (non zero), the block
* of code immediately following is executed,
* until the corresponding IFNOT or IFEND.
* If IFNOT is present, the following block of code
* is skipped until the corresponding IFEND.
* If the test result is FALSE (zero), the following
* block of code is skipped until the corresponding
* IFNOT or IFEND, and execution resumes there.
* (NOTE: Use of unusual forms IFNOT and IFEND
* is forced because TSC macro assembler already
* has ELSE and ENDIF mnemonics.)

* a) On compare

* Accepts logical operators EQ, NE, GE, LT,
* GT, LE with the same meaning as in the 6809
* assembler or IR FORTRAN IF instructions.
* <reg> can be any of the processor registers
* except DP, PC or CC. <addr> can use any of
* the legal addressing modes of the 6809
* (but an indexed address including a comma,
* plus or minus signs, should be in quotes).
* Optional parameter L signals a long branch.
* TYPICAL USE: IFRG B,NE,"0,X+",L
*                <block of code>
*              IFNOT
*               <block of code>
*              IFEND

IFRG    MACRO   <reg>,<cond>,<addr>,[L]
        PAT2T   &3
        IFN     BNP4,5
                LGT2T   &4
                RGT5T   &1
                CMP&1   &3
                OPT5T   &2,C
                P6SH    *-1-BR1LG
                ENDM

* b) On A or B register or memory TST

* Takes advantage of the 6809 TST
* instruction, which tests the contents
* of the A or B register or a memory byte
* and sets some flags accordingly.
* The only accepted conditions are EQ, NE,
* LT and GE.
* TYPICAL USE: IFTST A,LT
*                etc...

IFTST   MACRO   <reg>,<cond>,[L]
        PAT2T   &2
        IFN     BNP4,6
        LGT2T   &3
        RGT5T   &1,C
        IFN     BNR3G,3
        TST     &1
        OPT5T   &2,T
        P6SH    *-1-BR1LG
        ENDM

* c) On condition code register status

* Supposes that the needed test has
* already been made, and bases its choice
* on the state of one or several of the
* condition code register flags. This is
* the general form, where all possible
* conditions are accepted.
* TYPICAL USE: JSR FMS
*              IFCC NE,L
*               <error treatment>
*              IFEND
*               <execution continues>

IFCC    MACRO   <cond>,[L]
        PAT2T   &1
        IFN     BNP4,3
        LBT2T   &2
        OPT5T   &1
        P6SH    *-1-BR1LG
        ENDM

* END OF THE IF BLOCK

IFEND   MACRO
NT4DR   SET     *
        P6LL
        IFN     BRL26
        IF      (NT4DR-*-1)>127
        ERR     *'L' AFTER IF*
        ELSE
        FCB     NT4DR-*-1
        ENDIF
        ELSE
        FDB     NT4DR-*-2
        ENDIF
        ORG     NT4DR
        ENDM

* ELSE TYPE CONSTRUCT

IFNOT   MACRO   [L]
        LBT2T   &1
        IF      BR1LG
        LBRA    RP1LG
        ELSE
        BRA     *
        ENDIF
        IFEND
        P6SH    *-1-BR1LG
        ENDM
```

```
* LOOP STRUCTURES

* a) TEST AT THE BEGINNING

* Repeats a block of instructions as long
* as the test gives a TRUE result, with the
* same conditions as the IFRG structure.
* Test is made at the start, so that the
* block of instructions may not be executed
* at all.
* TYPICAL USE: WHILE A,NE,"0,-U"
*                <block of code>
*              ENDWHILE
WHILE   MACRO   <reg>,<cond>,<addr>,[L]
        PAT2T   &3
        IFN     BNP4,6
        LGT2T   &4
AD2PL   SET     *
        RGT5T   &1
        CMP&1   &3
        OPT5T   &2,C
        P&SH    *-1-BR1LG,0
        ENDM

* END OF WHILE LOOP
ENDWHILE MACRO
NT4DR   SET     *
        P&LL    0
        IFN     BRL2G
        IF      NT4DR-*+1>127
        ERR     *'L' AFTER WHILE*
        ELSE
        FCB     NT4DR+1-*
        ENDIF
        ELSE
        FDB     NT4DR+1-*
        ENDIF
        ORG     NT4DR
        IFN     BRL2G
        BRA     AD2PL
        ELSE
        LBRA    AD2PL
        ENDIF
        ENDM

* b) TEST AT THE END

* The simplest (and fastest) possible
* loop structure, requires only a test
* and branch instruction at the end.
* Follows the same rules as the WHILE
* loop, but for an important exception:
* condition is set and tested at the end of
* the block of code, so that the block must
* be executed at least once. The beginning
* REPEAT pseudo-instruction takes only one
* parameter, the optional L.
* TYPICAL USE: REPEAT L
*                <block of code>
*              UNTIL Y,GT,"S+*"
REPEAT  MACRO   [L]
        LGT2T   &1
        P&SH    *
        ENDM

UNTIL   MACRO   <reg>,<cond>,<addr>
        PAT2T   &3
        IFN     BNP4,19
NT4DR   SET     *
        P&LL
BR1LG   SET     BRL2G
JUSQ2   SET     *
        ORG     NT4DR
        RGT5T   &1
        CMP&1   &3
        OPT5T   &2,C
        ORG     *-1-BR1LG
        IFN     BR1LG
        IF      *-JUSQ2-1>127
        ERR     * 'L' AFTER UNTIL *
        ELSE
        FCB     JUSQ2-*-1
        ENDIF
        ELSE
        FDB     JUSQ2-*-2
        ENDIF
        ENDM

* c) COUNTING LOOP

* Executes a block of instructions while
* incrementing or decrementing a register
* until the register contents passes some
* preset limit. Equivalent to the BASIC
* FOR NEXT loop except that here the test
* is made at the beginning, so that the
* block of instructions may never be
* executed. Note that for practical coding
* reasons, the initialisation and limit of
* the register are set at the beginning, but
* the increment or decrement is defined
* only at the close of the loop.
* TYPICAL USE: FOR X,#5000,TEN
*                <block of code>
*              NEXT X,-5

FOR     MACRO   <reg>,<min>,<max>,[L]
        PAT2T   &3
        IFN     BNP4,11
        RGT5T   &1,M
        IFN     BNR3G,9
        LGT2T   &4
        LD&1    &2
AD2PL   SET     *
        CMP&1   &3
        IF      BR1LG
        LBHI    RP1LG
        ELSE
        BHI     *
        ENDIF
        P&SH    *-2-BR1LG,0
        ENDM

* END OF FOR LOOP

NEXT    MACRO   <reg>,<diff>
        PAT2T   &2
        IFN     BNP4,34
        RGT5T   &1,M
        IFN     BNR3G,32
        IF      BNR3G=2
        LEA&1   &2,&1
        ELSE
        IFC     &1,D
        ADD&1   #&2
        ELSE
        IF      &2=1
        INC&1
        ELSE
        IF      &2=-1
        DEC&1
        ELSE
        ADD&1   &2
        ENDIF
        ENDIF
        ENDIF
        ENDIF
NT4DR   SET     *
        P&LL    0
        IF      &2>$7FFF
        BLO     *
        ORG     *-1
        ELSE
        ORG     *+1
        ENDIF
        IFN     BRL2G
        IF      NT4DR-*+1>127
        ERR     *'L' AFTER FOR*
        ELSE
        FCB     NT4DR+1-*
        ENDIF
        ELSE
        FDB     NT4DR+1-*
        ENDIF
        ORG     NT4DR
        IFN     BRL2G
        BRA     AD2PL
        ELSE
        LBRA    AD2PL
        ENDIF
        ENDM

* MULTIPLE CHOICE STRUCTURE

* This is equivalent to the Pascal CASE
* structure, which allows you to compare
* a variable with several different possible
* values, and execute this or that section
* of code according to the result. Here, the
* base variable is one of the registers (same
* as in IFRG structure), and the only test
* made is for equality. There are four macros
* in the structure:
* CASE only determines which register is used.
* OF, repeated as many times as necessary,
* introduces one of the possible values and the
* corresponding block of code. The value may be
* defined using any addressing mode.
* OTHER is the default option, taking care of
* any case not covered in the OF blocks. It is
* optional.
* ENDCASE marks the end of the structure.
* TYPICAL USE: CASE A
*                OF #'+
*                 <code for add>
*                OF #'-
*                 <code for sub>
*                OF #'*
*                 <code for mult>
*                OF #'/
*                 <code for div>
*                OTHER
*                 <number or variable>
*                ENDCASE
CASE    MACRO   <reg>
RG2OD   SET     0
        IFC     A,&1,12
RG2OD   SET     1
        IFC     B,&1,10
RG2OD   SET     2
        IFC     D,&1,8
RG2OD   SET     3
        IFC     U,&1,6
RG2OD   SET     4
        IFC     X,&1,4
RG2OD   SET     5
        IFC     Y,&1,2
        ERR     * ILLEGAL REGISTER &1 *
        EXITM
RG2OD   SET     RG2OD+1
AD2PL   SET     0
BRL1G   SET     0
        P&SH    0,0
        ENDM

OF      MACRO   <addr>,[L]
        PAT2T   &1
        IFN     BNP4,29
        LGT2T   &2
NT4DR   SET     *
        P&LL    0
NT2DR   SET     *
        IFN     NT2DR,21
        IFN     AD2PL,6
        ORG     AD2PL
        IF      BRL2G
        FDB     NT4DR-*-2
        ELSE
        FCB     NT4DR-*-1
        ENDIF
        ORG     NT4DR
AD2PL   SET     *+1
        IF      BR1LG
        LBRA    RP1LG
        ELSE
        BRA     *
        ENDIF
NT4DR   SET     *
        ORG     NT2DR
        IF      BR1LG
        FDB     NT4DR-*-2
        ELSE
        FCB     NT4DR-*-1
        ENDIF
        ORG     NT4DR
        GINOD   &1
        P&SH    *-1-BR1LG,0
        ENDM
```

```
OTHER   MACRO   {L}
        LGT2T   &1
NT4DR   SET     *
        P&LL    Q
NT2DR   SET     *
        IFN     NT2DR,21
        IFN     AD2PL,6
        ORG     AD2PL
        IF      BRL2G
        FDB     NT4DR-*-2
        ELSE
        FCB     NT4DR-*-1
        ENDIF
        ORG     NT4DR
AD2PL   SET     *+1
        IF      BR1LG
        LBRA    BR1LG
        ELSE
        BRA     *
        ENDIF
NT4DR   SET     *
        ORG     NT2DR
        IF      BR1LG
        FDB     NT4DR-*-2
        ELSE
        FCB     NT4DR-*-1
        ENDIF
        ORG     NT4DR
        P&SH    0,Q
        ENDM

ENDCASE MACRO
NT4DR   SET     *
        P&LL    Q
NT2DR   SET     *
        IFN     NT2DR,5
        IF      BRL2G
        FDB     NT4DR-*-2
        ELSE
        FCB     NT4DR-*-1
        ENDIF
        ORG     AD2PL
        IF      BRL2G
        FDB     NT4DR-*-2
        ELSE
        FCB     NT4DR-*-1
        ENDIF
        ORG     NT4DR
        ENDM
```

```
* SEARCH COMMAND

* LOOKS THROUGH A FILE FOR A SERIES
* OF HEX VALUES

* YVES LECLERC, 12/2/82

* FLEX ROUTINES AND ADDRESSES

WARMS EQU $CD03
GETFIL EQU $CD2D
GETHEX EQU $CD42
OUTADR EQU $CD45
OUTHEX EQU $CD3C
PUTCHR EQU $CD18
SYSFCB EQU $C840
RPTERR EQU $CD3F
FMS EQU $D406
FMSCLS EQU $D403
PSTRNG EQU $CD1E
PCRLF EQU $CD24

* INCLUDE STRUCTURED MACROS

 OPT NOL
 LIB ASTRUCA
 OPT LIS

* PROGRAM BEGINS

 ORG $C100

FALST BRA TRUEST

* VARIABLES

LENGTH FCB 1    Lenght of byte list
SCTADR FDB 0    Address of present sector
CURBYT RMB 1    Byte addressed in sector
LSECT FCB 0     Remaining lenght of sector

* GET FILE NAME AND BYTE LIST

TRUEST LDX #SYSFCB
 JSR GETFIL
 IFCC CC,L          If no error in name
   LDY #BYTLST
   JSR GETHEX
   WHILE B,NE,#0  Get list of bytes to compare
     TFR X,D
     STB Y+
     JSR GETHEX
   ENDWHILE
   TFR Y,D          Compute lenght of byte list
   SUBD #BYTLST
   STB LENGTH
   JSR PCRLF
   LDA #1           Open file
   LDX #SYSFCB
   STA ,X
   JSR FMS

* START READING FILE

   IFCC EQ
     LDY #BYTLST    Get first byte
     JSR FMS
     IFRG A,EQ,#2   Check if binary file
       DEC 39,X
       LBSR BEGSCT    Take sector length
       JSR FMS
         BNE ERROR
     IFEND

* LOOK FOR A MATCH

     REPEAT
       BSR ENDTST
       IFRG A,EQ,Y    If first byte matches
         LDB LENGTH
         LDA 34,X   Note position
         STA CURBYT
         DECB       Is it a single byte?
         IFCC NE No, check rest of list
           LEAY 1,Y
           REPEAT
             JSR FMS
               BNE ERROR
             BSR ENDTST
             IFRG A,NE,Y+ If no match, jump
               BRA NOMTCH
             IFEND
             DECB
           UNTIL B,EQ,#0
         IFEND

* MATCH FOUND

         LDY #BYTLST
         LEAX 30,X
         LDD 0,X      Does it bridge two sectors?
         IFTST CURBYT,EQ
           DECB       Adjust sector address
           IFCC EQ
             DECA
           IFEND
         IFEND
         IFRG D,NE,SCTADR  If not same sector as
           STD SCTADR      last match, print sector
           JSR PCRLF       address.
           LDX #SCTADR
           JSR OUTADR
         IFEND
         LDA #'/    Print position of byte in sector
         JSR PUTCHR
         DEC CURBYT
         LDX #CURBYT
         JSR OUTHEX
         LDX #SYSFCB
       IFEND
     NOMTCH LDY #BYTLST
           JSR FMS    Get next byte of file
           IFCC NE
             CLRB       Flag "end of file"
           IFEND
         UNTIL B,EQ,#0
       IFEND
     ERROR BRA PRGEND

* BEGINNING OF BINARY SECTOR

     ENDTST IFTST LSECT,NE
         DEC LSECT
         IFTST LSECT,EQ If end of sector
           BSR BEGSCT Go to next
           JSR FMS And get first byte
           IFCC NE
             PULS Y
             BRA ERROR
           IFEND
         IFEND
       IFEND
       RTS

BEGSCT JSR FMS Pass loading address
 JSR FMS
 JSR FMS
 STA LSECT And note length
 RTS

PRGEND LDA 1,X
    IFRG A,NE,#8
      JSR RPTERR
    IFEND
    JSR FMSCLS
  IFEND  End of main program
  JMP WARMS

 BYTLST EQU *
  END FALST
```

* STANDARD PRE-NAMED LABEL EQUATES

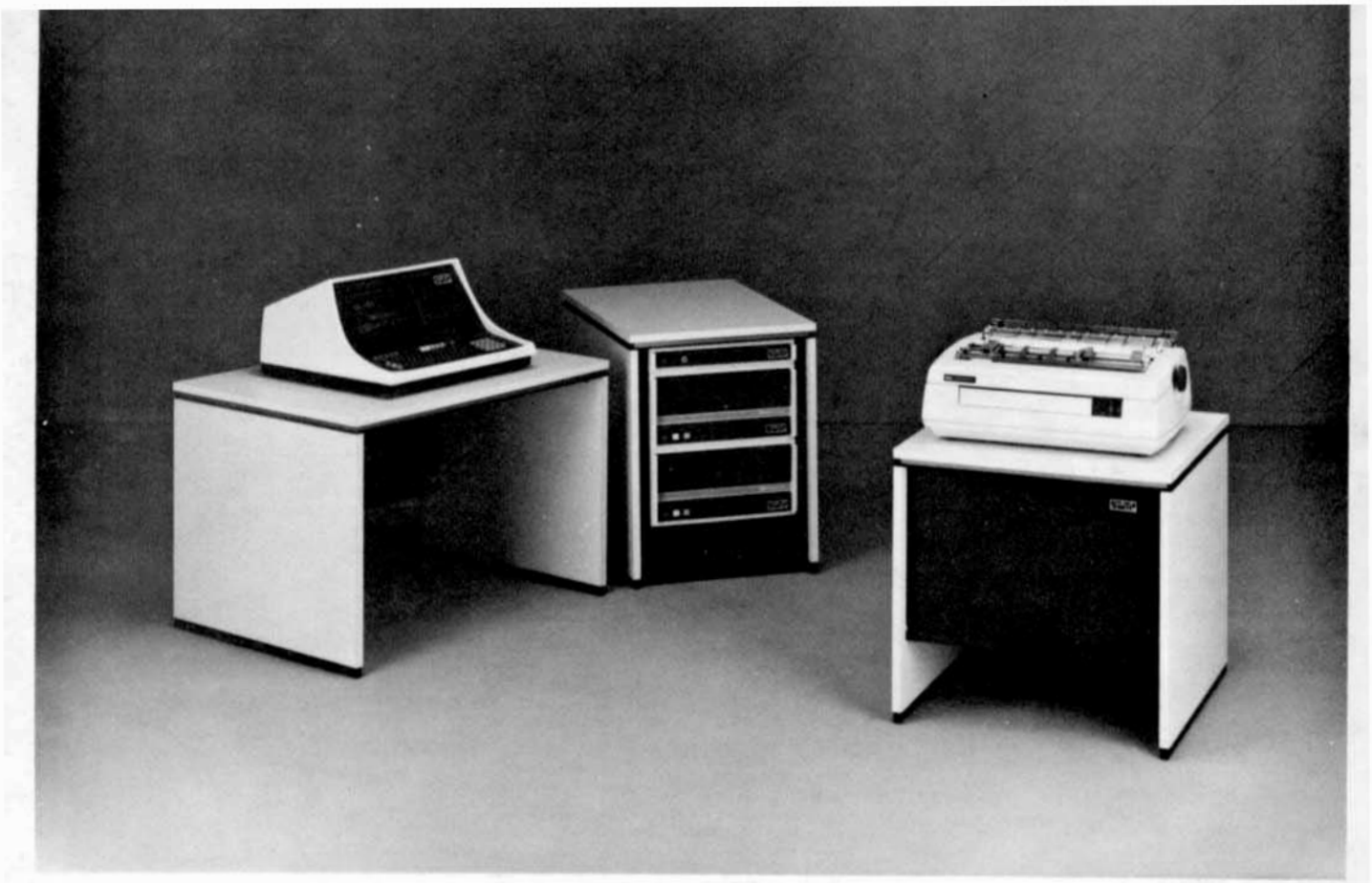
```
C840  SYSFCB  EQU   $C840
CD03  WARMS   EQU   $CD03
CD18  PUTCHR  EQU   $CD18
CD24  PCRLF   EQU   $CD24
CD2D  GETFIL  EQU   $CD2D
CD3C  OUTHEX  EQU   $CD3C
CD3F  RPTERR  EQU   $CD3F
CD42  GETHEX  EQU   $CD42
CD45  OUTADR  EQU   $CD45
D403  FMSCLS  EQU   $D403
D406  FMS     EQU   $D406
```

* EXTERNAL LABEL EQUATES

```
C1F2  LC1F2   EQU   $C1F2
```

```
:100                        ORG   $C100

:100 20  05       LC100     BRA   LC107
:102 01           LC102     FCB   $01
:103 00 00        LC103     FCB   $00,$00
:105 45           LC105     FCB   $45
:106 00           LC106     FCB   $00
:107 BE  C840     LC107     LDX   #SYSFCB
:10A BD  CD2D               JSR   GETFIL
:10D 1025 00DE              LBCS  LC1EF
:111 108E C1F2              LDY   #LC1F2
:115 BD  CD42               JSR   GETHEX
:118 C1  00       LC118     CMPB  #0
:11A 27  09                 BEQ   LC125
:11C 1F  10                 TFR   X,D
:11E E7  A0                 STB   0,Y+
:120 BD  CD42               JSR   GETHEX
:123 20  F3                 BRA   LC118
:125 1F  20       LC125     TFR   Y,D
:127 83  C1F2               SUBD  #$C1F2
:12A F7  C102               STB   LC102
:12D BD  CD24               JSR   PCRLF
```



THE COMPLETE BUSINESS SYSTEM

+Multiuser+Highly Expandable+Cost Effective

S+ THE CONCEPT

The S+ system is a modular computer system in which all portions of the hardware and software are designed to work together in the most efficient way possible. An S+ single user system with floppy disk storage is a competitive and cost effective entry level system. Unlike most other small computers being sold as "personal", or "small business" machines, the S+ system may be expanded to maximum capabilities using this same hardware and software. You cannot end up with a DEAD END system that cannot be expanded and whose software is not compatible with larger machines. A basic S+ system may be expanded to thirty-two users, a megabyte of main memory and hundreds of megabytes of hard disk storage by simply plugging in, or connecting the desired upgrade equipment.

TOTAL DESIGN—Hardware and Software

The S+ system is an integrated hardware and software design. The two complement and enhance each other in this system. The UniFLEX® operating system used in the S+ systems is patterned after the Bell Laboratories UNIX® operating system, one of the most admired and widely used operating systems in the world. Instead of being an afterthought, the software is part of the design of the S+ system. You can be sure that with this approach that all parts of the computer operate with maximum efficiency and cost effectiveness.

THE CENTRAL PROCESSOR

The basic S+ system is configured with 256K bytes of memory and can be expanded to more than 1 million bytes. An efficient and fast hardware memory management system is used to allocate the available memory among the users on a dynamic basis. As little as 8K bytes, or the entire memory—if needed—can be used by any individual user. This makes it possible to run very large programs on the system, but it also uses no more memory than necessary for a particular job. The increase in cost effectiveness of this system over crude and outdated bank switching arrangements is dramatic.

The central processor runs in both user and supervisor states. It can detect and reject a defective user program. It is impossible for a user program to go bad and stop the entire system, as can happen quite easily in less sophisticated systems.

Task switching is accomplished by use of a multiple map RAM memory, with sixty-four individual task maps. Each task can access from 4 to 64 K-bytes of memory. Multiple tasks may be used in programs that require more than 64K bytes of memory for execution. When a task is completed the memory is automatically released for other use.

SOFTWARE

The S+ operating system, UniFLEX® is a multiuser, multitasking operating system based on the UNIX® operating system that has been used for many years on Digital Equipment Corp. PDP-11 series minicomputers. It is considered one of the most sophisticated and "user friendly" operating systems available. Variations of UNIX® are rapidly becoming standard on mini and larger microcomputers.

A large variety of languages are available for use with the system. These include FORTRAN, COBOL, BASIC, and Pascal. Word processing packages are also available to give you full text processing capability on the system.

Applications programs are available in large quantities in many fields. This includes general business, medical, dental, veterinary, library and real estate management; plus others. Since the system is multiuser it can also be connected to cash registers to produce a point-ofsale terminal system combined with the computer. The possibilities for application of this system are endless.

THE I/O SYSTEM

The S+ system is totally interrupt driven. All terminal and printer I/O devices connect to an I/O bus separate from the main bus. Up to thirty-two separate devices may be connected to the I/O bus at any one time. If I/O activity is great enough to cause an unacceptable slowdown in system operation, a separate I/O processor can be installed in the system. This plug-in option removes all I/O handling overhead from the main processor and allows operation of up to thirty-two external devices at 9,600 baud. Without an integrated total design, as in the S+ system, it would become impractical to use a UNIX®type operating system in a situation with heavy terminal I/O activity.

DISK STORAGE

A wide range of disk storage capacity is available for the S+ system, from 2.5 M-byte floppy disks to an 80 M-byte Winchester and many sizes between. All disk controllers use direct memory access (DMA) type operations to maximize data transfer and to minimize overhead on the main processor. The Winchester disks also use intelligent controllers along with DMA transfers to preserve the performance that these type devices are capable of giving. Without this distributed intelligence the system performance would be greatly degraded. The UniFLEX®operating system is designed to work at maximum efficiency with this type disk system. The data transfer rates achieved by this combination rival those of large minicomputers.

COMMUNICATIONS

A high speed local network communications system is available to interconnect S+ systems. The VIABUS® network will allow communication between systems at data rates of over 400K baud. Such a system makes it possible to share data between local systems in an efficient and low-cost manner.

AVAILABLE SOON

Tape backup—20M-Byte in less than 15 minutes on a standard ¼ inch cartridge.

Mini-Wini—5 and 10 M-Byte Winchesters—5¼ inch package. Winchester performance, for smaller systems in a small package. UniFLEX® compatible design.

Large Capacity—190 and 340 M-Byte Winchesters, plus SMD cartridge drives.

UniFLEX is a registered trademark of Technical Systems Consultants, Inc.

UNIX is a registered trademark of Bell Labs.

VIABUS is a registered trademark of Southwest Technical Products Corporation.

```
:130 86   01          LDA   #1
:132 BE   CB40        LDX   #SYSFCB
:135 A7   84          STA   0,X
:137 BD   D406        JSR   FMS
:13A 26   7F          BNE   LC1BB
:13C 108E C1F2        LDY   #LC1F2
:140 BD   D406        JSR   FMS
:143 B1   02          CMPA  #2
:145 26   0B          BNE   LC152
:147 6A   8B 3B       DEC   59,X
:14A 17   0089        LBSR  LC1D6
:140 BD   D406        JSR   FMS
:150 26   69          BNE   LC1BB
:152 8D   69          BSR   LC1BD
:154 A1   A4          CMPA  0,Y
:156 26   55          BNE   LC1AD
:158 F6   C102        LDB   LC102

C15B A6   8B 22       LDA   34,X
C15E B7   C105        STA   LC105
C161 5A               DECB
C162 27   14          BEQ   LC17B
C164 31   21          LEAY  1,Y
C166 BD   D406  LC166 JSR   FMS
C169 26   50          BNE   LC1BB
C16B 8D   50          BSR   LC1BD
C16D A1   A0          CMPA  0,Y+
C16F 27   02          BEQ   LC173
C171 20   3A          BRA   LC1AD
C173 5A         LC173 DECB
C174 C1   00          CMPB  #0
C176 26   EE          BNE   LC166
C178 108E C1F2  LC178 LDY   #LC1F2
C17C 30   8B 1E       LEAX  30,X
C17F EC   84          LDD   0,X
C181 7D   C105        TST   LC105
C184 26   04          BNE   LC18A
C186 5A               DECB
C187 26   01          BNE   LC18A
C189 4A               DECA
C18A 10B3 C103  LC18A CMPD  LC103
C18E 27   0C          BEQ   LC19C
C190 FD   C103        STD   LC103

C193 BD   CD24        JSR   PCRLF
C196 BE   C103        LDX   #LC103
C199 BD   CD45        JSR   OUTADR
C19C 86   2F    LC19C LDA   #$2F
C19E BD   CD1B        JSR   PUTCHR
C1A1 7A   C105        DEC   LC105
C1A4 BE   C105        LDX   #LC105
C1A7 BD   CD3C        JSR   OUTHEX
C1AA BE   CB40        LDX   #SYSFCB
C1AD 108E C1F2  LC1AD LDY   #LC1F2
C1B1 BD   D406        JSR   FMS
C1B4 27   01          BEQ   LC1B7
C1B6 5F               CLRB
C1B7 C1   00    LC1B7 CMPB  #0
C1B9 26   97          BNE   LC152
C1BB 20   26    LC1BB BRA   LC1E3
C1BD 7D   C106  LC1BD TST   LC106
C1C0 27   13          BEQ   LC1D5
C1C2 7A   C106        DEC   LC106
C1C5 7D   C106        TST   LC106
C1C8 26   0B          BNE   LC1D5
C1CA 8D   0A          BSR   LC1D6
C1CC BD   D406        JSR   FMS
C1CF 27   04          BEQ   LC1D5
C1D1 35   20          PULS  Y
C1D3 20   E6          BRA   LC1BB
C1D5 39         LC1D5 RTS
C1D6 BD   D406  LC1D6 JSR   FMS
C1D9 BD   D406        JSR   FMS
C1DC BD   D406        JSR   FMS
C1DF B7   C106        STA   LC106
C1E2 39               RTS
C1E3 A6   01    LC1E3 LDA   1,X
C1E5 81   08          CMPA  #8
C1E7 27   03          BEQ   LC1EC

C1E9 BD   CD3F        JSR   RPTERR
C1EC 8D   D403  LC1EC JSR   FMSCLS
C1EF 7E   CD03  LC1EF JMP   WARMS

          C100        END   LC100
```

SMALL/SIMPLE EXEC-68KECB-68000

by Phil Nuclos
Motorola Inc., Semiconductor Products Sector

Executing two independent programs on the same hardware with in-line code is difficult. Executing more than two is very difficult. The ECBEXEC is a control program which greatly simplifies this task. Up to seven User programs may be run concurrently, but not simultaneously. The ECBEXEC controls the sequence and duration of execution and assures that each user program receives equal time if desired. Provisions are made to allow use of the interrupt vectors, and to start or stop other user programs.

Functional Description:

SMALL (2K hex bytes)

Up to 7 user programs may be run.

Allows use of 68K Autovectors plus any eight other 68K vectors.

Allows user programs to be deactivated or activated with or without delays.

Allows maximum of 1 sec execution time before termination.

Easily merged, linked or combined with use programs.

Relocatable

ALL user programs operate in the USER mode.

Theory of Operation:

The first job of the ECBEXEC on start up is system initialization. The executive stack is established. All active user program entries in the scan table (ie; PGMPC≠0), will have their stacks initialized. The exception vectors will be initialized and the timer will be initialized. No other hardware is initialized by the exec.

The ECBEXEC now begins to scan the table looking for programs to execute. The table start address is loaded, and the delay count of the first program is tested. If it is =0, the table pointer is incremented and the next entry checked. If it is ≠0, the program stack is fetched from the table and the machine contents are restored. An RTE is then executed to jump to the User's program. Since this is the first time this program has been run, the program counter in the table should have been pointing to the initialization routine for that program.

The initialization routine for this user program requests interrupt service (if necessary) from the exec. It also sets up any parameters needed later in the program, and then branches to the main program.

Execution continues in the user program until one of two things happens: 1) the program requests termination or 2) the program timer times out. The result is the same. The entire machine state is saved on the User stack, and the stack pointer is stored in the scan table.

The ECBEXEC then begins scanning for another scan table entry whose delay is zero. When it finds an entry whose delay is zero, it fetches the stack pointer from the scan table and restores the machine state. It then executes an RTE to jump to the User's program. If the end of table is reached, the Executive stack is restored, and the table start address is reloaded. The ECBEXEC then begins scanning from the beginning of the table.

The PIA timer is active during the entire time the exec is operational. The timer provides level 2 interrupts every 250 milliseconds. Two exec routines handle the timer. The interrupt handler, TINT, removes the interrupt and increments the TICKER and the program timer. It also checks to see if the program timer = 0. If it is, the interrupt handler begins the process to terminate the program. The TICKER contains the count since TIMER was active. The second timer routine is called simply TIMER. TIMER manages the delay counts for all scan table entries. TIMER is the first entry in the scan table. It gets the value of TICKER and then clears the TICKER. It then tests each entry in the scan table. Delay counts of zero or $FFFF (forever) are skipped, but all others are decremented by the saved value of TICKER. Each entry is tested and/or decremented

ed until the end of the table is reached. TIMER then requests termination with 0 delay.

All ECBEXEC requests are handled through TRAP #1. Four possible actions can be requested. The request number is determined by D7.B as shown in the table below:

D7.B	Action taken
00	Terminate this routine forever.
01	Terminate for delay specified (D6.W has delay count)
02	Activate a program (A4.L & A5.L have program name and D6.W has delay count)
03	Interrupt vector use Request (D6.B has interrupt vector and A5.L has service routine address)

The first two requests will not return to the calling program. The second two will return to the caller upon completion. In all cases of TRAP #1, if the requested action was performed, then D7.B will contain that request. But if the request was denied for any reason, D7.B will be CLEARED. It is important to check this in the user program!

The first three can be called anytime. The fourth, Interrupt Service Request (ISR), NEED only be called once during the initialization portion of the user program.

Since only 15 interrupt requests are available, repeated use of the ISR will result in refusal sooner or later.

Combining User Programs with the ECBEXEC:

The ECBEXEC must be loaded into RAM. Any means of getting it there is acceptable. It does not have to ORG at $800, but it is suggested. It is easier to move once all User programs have been added.

Load the User programs into Ram beginning at $1000 or just above or below the ECBEXEC if it is not at $800. An assembly listing of the User programs is necessary.

Some decision will have to be made as to placement in the scan table as this determines the order of execution. This problem is left completely up to the User.

Perform a memory modify ;L on the user program #1 entry in the scan table. If a different delay is desired, enter it. Now modify the Delay count. If the program should be executed immediately, enter $0000. Now modify the PGM counter entry. Enter the LWORD starting address of the user program #1 here. The next LWORD after PGMPC is the Program Stack location. Enter the User Stack address here. The User Stack MUST be at least $50 bytes in length for each user program. Repeat this process until all User programs are entered into the scan table.

It is suggested that a copy of the entire program (EXEC+USER) be saved at this point.

If a different execution address is desired, the whole program may be moved to wherever the user wants. Note however, that the address in the scan table are ABSOLUTE and must be modified for the program to run at any other address. The ECBEXEC itself is completely relocatable.

The program may be started by executing at the first location of the ECBEXEC. Execution will continue until RESET or an uninitialized interrupt occurs.

Interrupts for which no ISR was granted or which are unused in the operation of the EXEC are uninitialized interrupts. These interrupts all have the same effect on the system. The 68K status register is loaded with $2700 and the processor stops. If the ECBEXEC stops system operation this is usually the cause. Place an ISR BEFORE you expect interrupts to occur.

A Simple Example:

Two simple User programs were used in the debugging of the ECBEXEC. The scan table of the ECBEXEC and program listings appear below:

The two example programs are alike except that they call different interrupt vectors and turn each other on.

Examining the first in detail we see that it has an initialization section which is the first part executed. Note that this section will be executed only once as no other part of the program branches to it again. Initialization consists only of a request to use an interrupt vector. The vector number, $42, and service routine address, $10xx , are both specified. The request is then made to the ECBEXC. A test is made upon return to see if the request was granted but no action is taken. After initialization, a long delay loop is encountered. Execution will continue here until the loop finishes or program time-out occurs (program time out occurs first!). The ECBEXEC terminates this program and goes looking for another to activate. It finds only one other, the TIMER program, which it executes. Upon termination of TIMER it is USERPGM1's time to run again. Execution begins where it left off in the delay loop. The loop finishes and

```
MD $000 $01DI
001000    1C3C0042              MOVE.B   #66,D6
001004    4DFA1054              LEA.L    $00001054,A6
001008    1E3C0003              MOVE.B   #3,D7
00100C    4E41                  TRAP     #1
00100E    0C070003              CMP.B    #3,D7
001012    4E71                  NOP
001014    203C0001FFFF          MOVE.L   #131071,D0
00101A    5340                  SUBQ.W   #1,D0
00101C    66F6                  BNE.S    $001014
00101E    4246                  CLR.W    D6
001020    287C4A554E4B          MOVE.L   #1247104587,A4
001026    9BCD                  SUB.L    A5,A5
001028    1E3C0002              MOVE.B   #2,D7
00102C    4E41                  TRAP     #1
00102E    0C070002              CMP.B    #2,D7
001032    6606                  BNE.S    $00103A
001034    4207                  CLR.B    D7
001036    4E41                  TRAP     #1
001038    60DA                  BRA.S    $001014
00103A    287C5553434D          MOVE.L   #1431520589,A4
001040    2A7C50474D32          MOVE.L   #1346850098,A5
001046    3C3C0004              MOVE.W   #4,D6
00104A    1E3C0002              MOVE.B   #2,D7
00104E    4E41                  TRAP     #1
001050    60E2                  BRA.S    $001034
001052    FFFF                  DC.W     $FFFF
001054    4E73                  RTE
001056    FFFF                  DC.W     $FFFF

TUTOR  1.2 > MD 2000 $01DI
002000    1C3C0043              MOVE.B   #67,D6
002004    4DFA2054              LEA.L    $00002054,A6
002008    1E3C0003              MOVE.B   #3,D7
00200C    4E41                  TRAP     #1
00200E    0C070003              CMP.B    #3,D7
002012    4E71                  NOP
002014    203C0001FFFF          MOVE.L   #131071,D0
00201A    5340                  SUBQ.W   #1,D0
00201C    66F6                  BNE.S    $002014
00201E    4246                  CLR.W    D6
002020    287C4A554E4B          MOVE.L   #1247104587,A4
002026    9BCD                  SUB.L    A5,A5
002028    1E3C0002              MOVE.B   #2,D7
00202C    4E41                  TRAP     #1
00202E    0C070002              CMP.B    #2,D7
002032    6606                  BNE.S    $00203A
002034    4207                  CLR.B    D7
002036    4E41                  TRAP     #1
002038    60DA                  BRA.S    $002014
00203A    287C5553434D          MOVE.L   #1431520589,A4
002040    2A7C50474D31          MOVE.L   #1346850097,A5
002046    3C3C0004              MOVE.W   #4,D6
00204A    1E3C0002              MOVE.B   #2,D7
00204E    4E41                  TRAP     #1
002050    60E2                  BRA.S    $002034
002052    FFFF                  DC.W     $FFFF
002054    4E73                  RTE
002056    FFFF                  DC.W     $FFFF
```

request is made to activate a program called 'JUNK ', A look at the scan table shows that the ECBEXEC will return an error because 'JUNK ' does not exist. Fortunately, a test is made on D7.B to determine if the requested action was taken. Since the request was refused, an attempt to activate another program occurs. This time the request is to activate "USERPGM2" with a delay of 4 (about 1 sec). Since "USERPGM2" does exist, this request will be granted however, no check is made to be certain of it. The last job of this program is to terminate itself. It does so with a delay of forever. This means that this program will never become active until another program requests it to become active.

It is important to remember that the next time this program becomes active, execution will begin with the instruction following the TRAP instruction. In this case that instruction is a branch to the warm start (after initialization). Keep this in mind when writing your own programs.

The second example program is different from the first only in that it requests use of vector number $43, and turns on 'USERPGM1'. The effect of running these programs with the ECBEXEC is that they alternately activate each

other, and delay about 1/2 second between executing. Any time spent non-executing user programs is spent scanning in the ECBEXEC.

Any type of user program may be used with the ECBEXEC. However, a few suggestions will be made. Stay away from long delay loops or polling routines. Try interrupt driven I/O instead. Real time delays may be generated but watch the execution time of other programs. Choose the order of programs in the table with care. Position in the table is very important. It is possible to request use of the two interrupt vectors which the ECBEXEC uses during its operation. Doing so crashes the ECBEXEC.

Using TUTOR with the ECBEXEC:

```
MD 858 C0
000858    54 49 4D 45 52 20 20 20  00 00 00 00 00 00 00 00  TIMER   ........
000868    00 00 0E 28 00 00 0F D0  55 53 45 52 50 47 4D 31  ...(....USERPGM1
000878    00 00 00 00 00 00 00 00  00 00 10 00 00 00 20 00  ................
000888    55 53 45 52 50 47 4D 32  FF FF 00 00 00 00 00 00  USERPGM2........
000898    00 00 20 00 00 00 30 00  55 53 45 52 50 47 4D 33  .. ...0.USERPGM3
0008A8    FF FF 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0008B8    55 53 45 52 50 47 4D 34  FF FF 00 00 00 00 00 00  USERPGM4........
0008C8    00 00 00 00 00 00 00 00  55 53 45 52 50 47 4D 35  ........USERPGM5
0008D8    FF FF 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0008E8    55 53 45 52 50 47 4D 36  FF FF 00 00 00 00 00 00  USERPGM6........
0008F8    00 00 00 00 00 00 00 00  55 53 45 52 50 47 4D 37  ........USERPGM7
000908    FF FF 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
TUTOR  1.2 > MM 858;L
000858    54494D45 ?
00085C    52202020 ?
000860    00000000 ?
000864    00000000 ?
000868    00000E28 ?
00086C    00000FD0 ?
000870    55534552 ?
000874    50474D31 ?
000878    00000000 ?
00087C    00000000 ?
000880    00001000 ?
000884    00002000 ?
000888    55534552 ?
00088C    50474D32 ?
000890    FFFF0000 ?
000894    00000000 ?
000898    00002000 ?
00089C    00003000 ?
0008A0    55534552 ?
0008A4    50474D33 ?
0008A8    FFFF0000 ?
0008AC    00000000 ?
0008B0    00000000 ?
0008B4    00000000 ?
0008B8    55534552 ?
0008BC    50474D34 ?
0008C0    FFFF0000 ?
0008C4    00000000 ?
0008C8    00000000 ?
0008CC    00000000 ?
0008D0    55534552 ?
0008D4    50474D35 ?
0008D8    FFFF0000 ?
0008DC    00000000 ?
0008E0    00000000 ?
0008E4    00000000 ?
0008E8    55534552 ?
0008EC    50474D36 ?
0008F0    FFFF0000 ?
0008F4    00000000 ?
0008F8    00000000 ?
0008FC    00000000 ?
000900    55534552 ?
000904    50474D37 ?
000908    FFFF0000 ?
00090C    00000000 ?
000910    00000000 ?
000914    00000000 ?.
```

It is possible to use TUTOR to debug user programs running under the ECBEXEC. To do so you must place NOPs over the BSRs to EVINIT and TINIT at the beginning of ECBEXEC. You must then set up the TRAP #1 vector with the address of UTRAP1. The operation of ECBEXEC and the user programs may then be observed with TUTOR. Note however, that the PI/T is NOT active and real-time delays will not work.

ECBEXEC is designed to help the student or hobbyist execute 68000 programs on the 68KECB. It provides the necessary controls to allow multi-tasking of programs and real-time delays before execution. Although many EXECs exist for the 68000, few are simple to use or small enough to fit the 68KECB.

```
MOTOROLA M68000 ASM VERSION 1.30S ... .EXEC ... 08/02/82 ...
[illegible]
```

```
[illegible]
...  SCAN TABLE OFFSET EQUATES
[illegible]
...  PROGRAM TIMEOUT TERMINATION ROUTINE
[illegible]
...  FUNCTION
[illegible]
```

REGULATOR PROBLEMS

by Peter A. Stark

The power supplies in S-50 systems deliver unregulated power to the motherboard; each board in the system then has its own voltage regulators which reduce the supply voltage to the required value and regulate it.

On some boards these regulators consist of a discrete circuit with one or more transistors, but in most cases boards use three-terminal regulator ICs such as the 7805 for +5 volts, the 7812 for +12 volts, and the 7912 for -12 volts.

I have several times run across three-terminal regulators which oscillate and produce strange looking output voltages which disrupt the operation of the board and, in some cases, the entire system. In most cases, these oscillations can be stopped by bypassing both the input and the output of the regulator with 0.1 uf capacitors, as close to the regulator as possible. In some cases larger capacitors may be needed, and perhaps the best method is to use tantalum (not electrolytic) capacitors of 1 to 10 uf.

Most plug-in boards have such capacitors near the regulators, but some do not. I have run across the problem with some DSD 16K memory boards, which have four 7805 regulators, none of which is bypassed on the input. In some systems these boards will simply not work until bypass capacitors are added.

It might therefore be a good idea to look at each three-terminal regulator in your system to see whether it is properly bypassed on both input and output; if not, then soldering small capacitors on the bottom of the board, right at the regulator pins, might be an excellent idea.

Next time you have the motherboard taken out of its case, it might also be a good idea to add a few bypass capacitors on the bottom, between the +8-volt unregulated line and the ground bus right next to it, up front in the 50-pin section as well as in the 30-pin section in the rear.

If you have an extension card (which plugs in between a board being tested and the motherboard to bring a board up above the others so you can get to it for testing), a few bypass capacitors on the extender board might also be worthwhile.

BIT Bucket

word's worth

P.O. Box 29054
Dallas, Texas 75278
(214) 321 9285

February, 1983

An Open Letter to a Thief:

I had on several occasions heard rumors that there was a relatively large 6809 club which was notorious for its software theft. In fact, rumor had it that this club even owned a photocopy machine which was available at the meetings for the purpose of copying documentation.

A few weeks ago, another rumor came up. This time, the grapevine informed me that the members of this particular club had taken up a collection at one of their meetings in order to buy Middle-C, a proprietary Small-C compiler of which I am the sole licensed distributor. Apparently, it was a major issue as to just who was going to put his name on the order. Obviously, they intend to spread this proprietary program around. I hear that they even put that sort of thing on their bulletin board.

So what can I do about it? Quite probably nothing. At least not yet.

Your order arrived within a few days of the time I expected it. I spent some time discussing with my partners about just how we should react to this problem. We decided to go ahead and fill it, and just grit our teeth in the knowledge that we are being ripped off.

For the first time since I started this project, I felt that I really wanted to quit the business. I had hoped that my very liberal upgrade policy, along with the lowest prices I could manage, and the sincere effort to provide my customers with good support would somehow cause me to eventually be spared from the thieves. Alas, 'tis not so.

So far, I have actually lost quite a bit of money on selling C compilers. The cost of advertising has more than eaten up the proceeds, especially in the beginning, when I was trying to place ads all over the place. Instead of just '68'. I just hoped that I could establish a good name and expand with other products, perhaps even a true public domain library. Now I am in the position of being ripped off, while I can do nothing but stand by and watch you do it. Maybe it wouldn't have felt so bad if your club hadn't been so damned open and blatant about it. I have never stolen anything from you.

Naturally, I can't prove anything, yet. You know who you are. If it makes any difference to you, I know who you are, also. In case you haven't guessed, I don't like you very much. But, until I get the proof I am looking for, you will be treated no differently from my other customers. You will get the same deal on updates. You will get the same support, the same prices. The only difference will be the little curse I pronounce over anything I send you.

Sincerely,

Howard Lee Harkness

Editor's Note: This is no isolated happening. Unlike many other groups, ours has in the past not been known as software 'copiers'. However, I have from time to time heard complaints from software vendors, of this sort of thing happening. In one particular case a software vendor **did** start legal proceedings in a situation somewhat like this. The case never came to court as the group 'attempted' a mass recall of the copied software. Also the guilty individuals were known and in two cases their employers severed their employment, due to possible adverse publicity (they were in the computer business). How do I know? I was a technical consultant, hired by the complaining company, to determine if the software copied was actually the 'real thing'. It was.

Since the advent of the color computer I have seen an upswing in this sort of illegal practice. In casual talk to some vendors it seems that they are becoming more alarmed about software thieves. It seems that the expense is sky-high to prosecute such a case. However, as has been done in other trades, a mutual cooperative fund could be set up, by all participating software vendors, this to cover most or all of a large part of the cost in a few well documented instances. A good **stiff** judgement would certainly slow down this practice. It was noted by the attorneys in the above case that if a club was involved, and the officers had knowledge or even should have had knowledge, of a thing such as this going on within the club, they also could (as well as the entire membership) be held **personally** libel for any damages awarded by court action. **Now that can get pretty expensive for even the innocent.** Being a member of most clubs also entails a lot of responsibilities. And sure could flatten the pocket book in a real hurry. So if you are a member of a club or group that steals (copies) copyrighted software, you are in line to have your wallet depleted.

I constantly hear complaints about software thieves. However, if the offended parties don't do anything, then there is not much can or will be done. You will just continue to be robbed and have to smile about it. I will do all we (68 MICRO JOURNAL) can to help stop this from happening. But it takes more, it takes a cooperative effort on the part of **all** software vendors or a very expensive action on the part of any particular one.

Even you hardware manufactures have a role in this also. The day has arrived that it takes more than just a computer and allied hardware to hold position in the marketplace. Without **good** software your days are numbered. I don't care who you are or how long you have been 'in the business'. If software becomes so expensive that **your** customers can no longer afford it, or you competitors can provide better and less expensive software, you are in big trouble!!!!!!!!!!!!

68 MICRO JOURNAL is planning to host a software and hardware manufacturers and dealers get together. Not a club or organization, but a getting together and trying to iron out some of our problems. Just because another company is your competitor does not mean that you do not have some of the same problems. No users or non-commercial types will be invited or allowed. This is just an opportunity to share **common** thoughts and discuss **common** problems. If you desire to participate please let me know within the next 60 days (by July 1st).

So I think we all should realize that if we **steal or copy** (little difference if we give it to someone who has not paid for the right to use it) it will someday be either be too expensive or no longer available.

It's up to you!

DMW - - -

DIGITAL RESEARCH CP/M-68K

Motorola has reached an agreement to sell the Digital Research CP/M-68K operating system for it's M68000 based EXORmacs™.

Also we were informed that LSI, a past advertiser in 68 MICRO JOURNAL™ hes also reached an agreement with Digital Research to license CP/M-68K for their new 68000 board for the Standard S50 Bus. The LSI Standard S50 Bus 68000 CPU card(s) with CP/M-68K will be advertised soon. The availability of other software such as CIS COBOL™ for the 68000 will allow many popular 8 and 16 bit microcomputer programs recompile access, bringing to the Standerd S50 Bus a wealth of time proven applications software.

CP/M68K is actuelly implemented in 'C' and is shipped with a complete C run-time package. Estimated cost is in the range of $350.00. It mainfains file compatibility with all Digital Research operating systems. Files from other 8 and 16 bit microcomputers running these systems can be transported with no conversion to 68000 based machines running CP/M68K.

In addition to Motorola and LSI there are other popular Standard S50 Bus manufacturers who will be offering their up-grade to 68000 hardware packages with this system standard. We will keep you informed as they near release.

- - -

GIMIX INC 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

PRESS RELEASE

GIMIX TO INTRODUCE NEW GMX III 6809 SYSTEM
at NCC Booth #P7946/8

GIMIX' new GMX III 6809 system features the new GMX III CPU board, OS-9 GMX III multi-user, multi-tasking operating system, and intelligent I/O processor board. The CPU board features high-speed memory to memory DMA transfers and automatic task switching on interrupts and operating system calls for bigger system throughput, and a Time-of-Day clock with battery backup. To prevent system crashes caused by errors in individual users programs, the system has fully protected user modes with illegal instruction and out of range memory reference trapping and write protection.

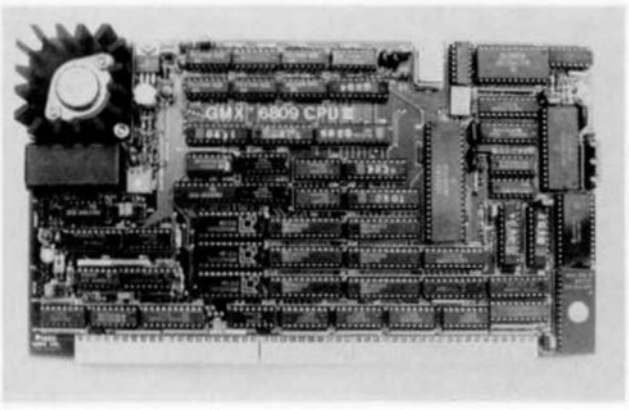
The system hardware also includes 256K bytes of static RAM, a 40M Byte (unformatted) 5.25" Winchester hard disk, a 1M byte (unformatted) 5.25" floppy disk, and 3 RS-232C I/O ports. Power is provided by a constant-voltage, ferro-resonant supply with sufficient reserve capacity to support a fully expanded system. Hardware options include memory expansion to 1 megabyte, non-volatile battery-backup RAM, additional mass storage capacity, and I/O ports for additional terminals and peripherals.

The UNIX-like OS-9 operating system includes the debugger, editor and assembler. Available software includes BASIC09, PASCAL, COBOL and C.

The base price for a 3 user system is $9,998.89. Delivery is from stock to 30 days A.R.O. Export models are available.

For further information contact: Richard Don at (312) 927-5510

GIMIX and GMX are trademarks of GIMIX Inc. Chicago, IL. OS-9 and BASIC09 are trademarks of Microware Systems Corp. Des Moines, IA



COMPUTER SYSTEMS CENTER
OF ST. LOUIS, INC.
13461 OLIVE BLVD.
CHESTERFIELD, MO. 63017
(314) 576-6020

March 28, 1983

PRESS RELEASE

Computer Systems Center announces the introduction of DYNASHARE for 6809 FLEX computer systems.

DYNASHARE, as its name implies, allows true "time-sharing" operation under the popular FLEX operating system. In addition to multi-user operation, DYNASHARE also allows each user to run two simultaneous jobs (multi-tasking).

The initial release of DYNASHARE is for SWTPC S/09 computers. But versions will also be available for other popular extended-memory systems, such as NCLIX and GIMIX. A minimum of 128K of RAM will be required in all versions.

DYNASHARE requires 64K of RAM for each active task: thus a 256K system could allow four-foreground/one-background operation or two simultaneous foreground-only operation of four terminals. Additional memory is supported to 4096K, depending on system hardware.

DYNASHARE's multi-tasking feature may be used even on single-user systems. For example, while in EDIT, you can list another file or examine a directory. Or you might look up an item in a data base even while a sort is in progress!

DYNASHARE also provides some fringe benefits that will be greatly appreciated by FLEX users, including type-ahead, command line editing, and instant response to "escape".

DYNASHARE is not intended as competition to UNIFLEX. It does not improve on the speed of FLEX, and does not offer password protection or other niceties of a full-blown multi-user system. What DYNASHARE does do is give FLEX users a low-cost way to use existing software in a multi-user, multi-tasking environment.

DYNASHARE is in customer-acceptance testing now. It will be available for sale on May 1, 1983. Price is $200 per copy; dealer inquiries invited.

FLEX and UNIFLEX are trademarks of Technical Systems Consultants, Inc.
DYNASHARE is a trademark of Computer Systems Center.

F & D Associates
1210 TODD ROAD
NEW PLYMOUTH, OHIO 45654

May 1, 1983

NEW PRODUCTS ANNOUNCEMENT

F & D Associates is pleased to announce the availability of the RDC-1, a disk controller card for the Color Computer. The board was developed using experience gained with our successful MDC-1 and MDC-1 disk controller boards for the S50 bus.

The RDC-1 uses either 1793 or 179) type FDC chips and the new SMC92168 data separator. This chip is less expensive than popular chip set separators and is all digital. The write precomp circuit on the RDC-1 is also all digital so no board adjustments are required.

The board is the same size as the Radio Shack controller and is completely compatible with it when the Radio Shack Disk Extended Basic rom is used. The board accepts either a 24 or 28 pin ROM or EPROM. EPROMs from 2716 thru 27128 can be used giving from 2k to 16k of on board memory.

We are developing a software package to allow running FLEX* without the need for Extended Basic and Disk Extended Basic. FLEX* will be booted using our FASBUG-C monitor. The RDC-1 is compatible with the FLEX* conversions developed by others when running with Disk Extended Basic.

Price of RDC-1 is $48.50 plus $3 shipping/handling. The price includes bare board, documentation package, and SMC92168 chip. We currently have available accessories such as a plastic case for the board, FDC1793 controller chips, 8 Disk Extended Basic ROM, and of course FASBUG-C. See our ads for prices.

We expect to be able to arrange for assembly of the board for people who do not feel qualified to assemble a board of this type.

David S. Weeks, President
F & D Associates
1210 Todd Road
New Plymouth, OH 45654
814 [illegible]

*FLEX is a trademark of Technical Systems Consultants, Inc.

DON MEITZENKAMP
RURAL ROUTE 1
LYONS, NEBRASKA 68038

I AM LOOKING FOR AN ACCOUNTING PROGRAM (WITH TEXTFILE STORAGE) IN TRS-80 EXTENDED BASIC FOR THE COLOR COMPUTER. IF YOU KNOW WHERE I COULD OBTAIN SUCH A PROGRAM IT WOULD BE GREATLY APPRECIATED IF YOU COULD LET ME IN ON THE SECRET.

SINCERELY,

DON'S PHOTO

DON MEITZENKAMP
OWNER OF DON'S PHOTO

Gentlemen:

I would like a 1 year subscription to the 68 MICRO JOURNAL (check enclosed). I also want to compliment you on the quality of the software reviews. After reading the review of the Dynamite disassembler (this was done some time ago) I finally bought a copy. It operated as advertised and the documentation was superb. My first use was in modifying the FLEX09 Catalog and copy utilities to display the file creation dates and to preserve them during disk transfer. With a good disassembler the job was trivial.

Yours,

Frank J Wilson
2468 Elden St., Apt J
Costa Mesa, Cal 92627

Dear Mr. Williams:

Thanks for publishing my article, "Flex Based Flexible Print System". The first news I had that you had published it in the March issue was a telephone call from a reader from Georgia who called to talk about how he could adapt the ideas in the article to his computer. It is satisfying to be able to help someone improve their FLEX system.

BUT, there is a problem with the article as printed. Your paste-up person left out two lines. They are the lines at $A23C to $A23F in SETPRINT.CMD. The missing lines are:

```
A23C B6 A1 05     LDAA TEMP+1
A23F 39           RTS
```

I would appreciate it if you would print this correction in the next available issue of "68" Micro.

For the record, this error was found first by Wesley Moffett of Saskatchewan, Canada.

Sincerely,

Kenneth Dreyler

Kenneth Dreyler

GIMIX inc. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312)927-5510 • TWX 910-221-4055

PRESS RELEASE

GIMIX introduces the GMX III 6809 CPU board and OS-9 GMX III. The new CPU board is an advanced design, specifically intended for use with multi-user, multi-tasking operating systems. OS-9 GMX III is an enhanced OS-9 Level II that takes full advantage of the features of the new CPU board. The price for the combination of CPU board and software is $1000.00

Built on a multi-layer (6) circuit board and utilizing high-speed, high-density logic, the GMX III 6809 CPU board enhances the performance of the 2 MHz 6809 by providing such features as high-speed (1 byte/microsecond) DMA block transfers from memory to memory or between memory and I/O devices (such as the GIMIX Intelligent 3 Port Serial Interface) and advanced memory management with 2K segments and segment attributes. The board automatically arbitrates DMA contention between the on board DMA and external DMA devices such as disk controllers. The 2K memory segments allow more efficient memory usage. The segment attributes allow the trapping of out-of-range memory references (to protect one user's or task's memory from being accessed by another), write protection (to protect shareable data and programs from modification which could affect the entire system), and a hardware single step function for software debugging (on a individual user basis without affecting other users or tasks).

The board prevents the execution of certain illegal instructions from crashing the system by monitoring interrupts to the 6809 and its response to them (these instructions cause the 6809 to lock up in a state in which it does not respond to any interrupts and must be reset). If the processor does not respond to an interrupt within a specific time (128 clock cycles) the board resets the 6809 (other devices on the bus are not reset) and executes a special reset vector. The system can then clean down the offending task and resume normal operation (other active tasks are not affected). This also limits the length of time that interrupts can remain masked by a user, preventing users from escaping the system from task switching and monopolizing other users.

To further protect the system from the users, the CPU board supports separate user and system "states" with automatic switching to the system state in response to interrupts and system (SWI) calls. Certain functions and memory areas can only be accessed in the system state, preventing unauthorized accesses.

Also included on the new CPU are an improved full function time-of-day clock (MC146818) with year and automatic leap year/daylight savings time correction, and a 2K battery backed RAM; both with battery backup standard. To provide precision timing functions, a 6840 PTM with a separate 500 KHz precision (.002%) time base oscillator is included. The oscillator is easily user replaceable to provide other time base frequencies (750 KHz max.). The single EPROM socket will accept 2K, 4K or 8K EPROMs, with a maximum of 4K mapped into the system address space at any one time. Software switching is implemented by selecting the upper or lower half of an 8K EPROM under hardware or software control.

By taking advantage of the features of the GMX III CPU, OS-9 GMX III is faster, more memory efficient, and a more secure multi-user/multi-tasking operating system than OS-9 GMX II, from which it is derived, while retaining complete software compatibility. Throughput is enhanced by the memory to memory DMA and the automatic task switching, while the memory attributes and illegal instruction trapping protect the system and individual users from each other. Sharable system modules in RAM are write protected to prevent tampering. Memory mapping in 2K segments and the ability to load modules in non-contiguous RAM provide more efficient memory utilization. Each task can be allocated a full 64K of RAM, with no operating system overhead in the tasks address space. Future plans for OS-9 GMX III include an optional hardware single stepping debugger.

Dear Mr. Williams:

I am writing this letter to announce the existence of a new 6809 based bulletin board system on the West Coast, and also to compliment you on a fine magazine!

The bulletin board, called 'BIT BUCKET', is located in Portland, Oregon. It currently runs on my homebuilt 6809 system, under FLEX. The system is written in TSC XBASIC, which has been modified to handle some of the peculiarities of remote use. The system runs with 3 Shugart 5 1/4" SSSD drives, and a custom built electronic disk drive.

The purpose of the system is to serve as a general purpose bulletin board. Currently, users may read and leave messages on the system. Also available are information files (other BB systems, FLEX and OS-9 notes, etc.). Planned improvements/additions are one way private mail (from System Operator to a user), system Help files, and possibly a downloadable set of FLEX utility and other programs.

The system runs 24 hours a day, with interruptions occasionally for software upgrades, backups, and other maintainance. The system officially started operating on January 1, 1983, and has been very successful so far. In a shade over 2 months of operation, there have been over 2100 callers, with quite a few from outside the local dialing area.

The system phone number is (503) 761-6345. Data rate is 300 baud only. Users are encouraged to leave comments or suggestions as to improvements, complaints, etc. concerning the system. These messages should be addressed to "System Operator" to assure that I see them.

I hope to hear from a lot of '68 Micro Journal readers!

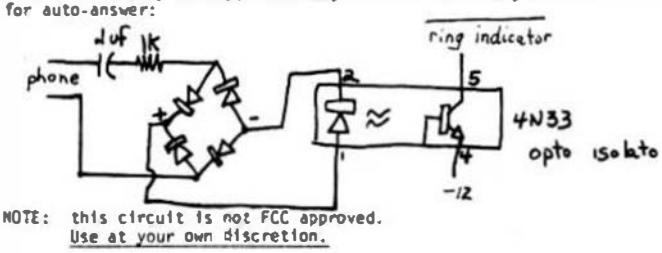
Rick Bensene

Rick Bensene
1815 N.E. 148th
Portland, Oregon 97230

Paul Burega
1 Pleasant Bay
Winnipeg, Manitoba
Canada R2K 0C9

Dear Don:

Enclosed find a modem program which I wrote for FLEX 9. I am using this program to give my 6809 access to BBS's and time sharing systems. I am using the modem board from Data Systems 68, which I think is just super. The board construction is of high quality, solder masked & silk screened, and it went together easily and worked first time. With the help of a friend (J. Gary Mills), I put together the following circuit for auto-answer:



NOTE: this circuit is not FCC approved.
Use at your own discretion.

This program can also be used with any modem which is connected to a serial port. The eQuates at the beginning of the program can be changed to allow for a different port address, as well as the ACIA initialization codes. The terminal I/O is handled from FLEX, so any terminal which works with FLEX will work.

When the program is called up, it asks for a file name to store the data. One must be given even if there is no wish to store the session. The user can then dial up another computer and all the session will be saved in memory. If the user does not wish to save it on disk, then the linefeed code will delete it and return to FLEX. Otherwise, the back-quote will cause the memory buffer to be saved on disk. Both of these characters are

changeable in the equates. You will then be requested to enter another file name to continue. Hitting just a carriage return will return you to FLEX. Once the session is saved on disk, it may be listed out or printed.

Sincerely,

Paul Burega

Paul Burega

```
                    OPT     PAG

                *
                * This program allows a FLEX 9 user to dial up
                * a BBS (bulletin board) or timesharing system and
                * download files.  Some editing of the files
                * may be necessary to eliminate unwanted text
                * at the beginning and the end of the download.
                *
                *  This program will first prompt for a file name
                *  After a response, it will open that file
                *  for writing.   It then will send a
                *  carriage return out thru the port
                *  (not to the system terminal).
                *  It then proceeds to transfer data between the
                *  port and the terminal.  Any data coming from
                *  the port is saved in memory.
                *  Upon typing backward Quote, all the contents
                *  of memory will be dumped to the file.
                *  If a LF (line feed) is typed, the program
                *  omits and none of the data is written out to the file.
                *  -- after writing the file, the user is
                *     prompted for another file name, which he
                *     must give.  To exit the program give a
                *     file called, then after it is opened,
                *     type a linefeed to delete it.
                *
                * When the memory buffer becomes full, a
                * control S is output to the modem port
                * to cause the remote computer to stop
                * sending data.  If your dial-up computer
                * uses a different code, change the line
                * which has the control S.  To restart
                * the incoming data from the remote
                * computer, a space is sent out.  If a
                * different character must be sent,
                * change that as well.
                *
F7EC  ACIADR    EQU     $F7EC       address of modem ACIA
F7ED  ACIADA    EQU     ACIADR+1
0060  BACKQT    EQU     '`          save buffer character
000A  LINEFD    EQU     $0A         program exit character
000A  ACIATP    EQU     $0A         acia init value,
                *                     7 data bits, even parity
                *                     use also for 8 data, not parity
                *                     for divide by 16 instead of 64
                *                     use $15.
                *
CD03  WARMS     EQU     $CD03
CD33  BETEXT    EQU     $CD33
CD1E  PSTRNG    EQU     $CD1E
CD18  PUTCHR    EQU     $CD18
CC2B  MEMEND    EQU     $CC2B
D406  FMS       EQU     $D406
CD3F  APTERR    EQU     $CD3F
CD2D  GETFIL    EQU     $CD2D
CD4E  STAT      EQU     $CD4E
CD09  INCH      EQU     $CD09

                CD1B  INBUFF  EQU   $CD1B

0100                    ORG     $100
0100 B6 03      MAIN    LDA     #3          RESET ACIA
0102 B7 F7EC            STA     ACIADR
0105 B6 0A              LDA     #ACIATP     get acia attributes
0107 B7 F7EC            STA     ACIADR      and set up acia
010A BE CC2B            LDX     MEMEND      Get last memory location
010D 30 89 FD00         LEAX    -$300,X     leave some free space
0111 AF 8D 0003         STX     MEMMA1,PCR  save end of buffer location
                *
                * get a file name
                *
0115 30 8D 0080 AGAIN   LEAX    MSG1,PCR    prompt system terminal
0119 BD CD1E            JSR     PSTRNG      for file name
011C BD CD1B            JSR     INBUFF      go get the characters
011F B6 CD80            LDA     $CD80       get first char of line buffer
0122 81 0D              CMPA    #$0D        check if C/R
0124 27 2B              BEQ     ERROR1      exit if so
0126 30 8D 0080         LEAX    FCB,PCR     point to file control block
012A BD CD2D            JSR     GETFIL      and get that file name
012D 25 1C              BCS     ERROR       exit if error
                *
012F B6 01              LDA     #1          set file for .TXT
0131 BD CD33            JSR     BETEXT
0134 86 02              LDA     #2          open for write
0136 A7 84              STA     ,X
0138 BD D406            JSR     FMS         and let FLEX open file
013B 26 0E              BNE     ERROR       exit if error
                *
013D 31 8D 01E9         LEAY    START,PCR   point to start of message buffer
0141 86 0D              LDA     #$0D        print out a carriage return
0143 BD 1F              BSR     OUTCR
0145 BD 0A      LOOP    BSR     CHTERM      check terminal for input
0147 BD 20              BSR     CHMOD       check status for input
0149 20 FA              BRA     LOOP        continue forever

014B BD CD3F    ERROR   JSR     APTERR      process errors
014E 7E CD03    ERROR1  JMP     WARMS       and the exit

                0151  CHTERM  EQU   *
                * check terminal for input
                *
0151 BD CD4E            JSR     STAT        check terminal status
0154 27 1A              BEQ     RET         do nothing if no input
0156 BD CD09            JSR     INCH        go get that character
0159 81 60              CMPA    #BACKQT
015B 27 4D              BEQ     DONE        if so, were done
015D 81 0A              CMPA    #LINEFD     LF READS QUIT
015F 26 03              BNE     OUTCR
0161 7E CD03            JMP     WARMS       RETURN TO FLEX

                0164  OUTCR  EQU   *
                * output a character to the modem

0164 B6 F7EC    CHI     LDA     ACIASR      check status first to see
0167 85 02              BITA    #2          if ready to send another char
0169 27 F9              BEQ     CHI         loop until ready
016B 35 02              PULS    A
016D B7 F7ED            STA     ACIADR
0170 39         RET     RTS

                0171  CHMOD  EQU   *
                * check modem for incoming characters

0171 B6 F7EC            LDA     ACIASR      check status
0174 84 01              ANDA    #1          for incoming character
0176 27 FB              BEQ     RET         if not, then leave
0178 B6 F7ED            LDA     ACIADR      otherwise, get that character
017B 84 7F              ANDA    #$7F        mask off parity bit
017D 81 0A              CMPA    #$0A        don't save linefeeds
017F 27 02              BEQ     NOST
0181 A7 A0              STA     ,Y+         otherwise, save it in memory
0183 BD CD18    NOST    JSR     PUTCHR      and output to the terminal
0186 10AC 8D 0058       CMPY    MEMMA1,PCR  check to see if near
018A 27 01              BEQ     DUMPER      end of memory
018D 39                 RTS

018E AE 8D 0056 DUMPER  LDX     MEMMA1,PCR  adjust memory end to prevent
0192 30 05              LEAX    5,X         being called again
0194 AF 8D 0050         STX     MEMMA1,PCR
                *
                *****************************************
                *
                * output control S to stop incoming data
                *
                *****************************************
                *
0198 86 13              LDA     #$13        control S
019A BD CB              BSR     OUTCR       output control S
019C BD D3              BSR     CHMOD       check for any remaining chars
019E 8D 18              BSR     DUMBUF      dump the buffer
01A0 86 20              LDA     #$20        re-start sending with space
01A2 8D CO              BSR     OUTCR
01A4 10AE 8D 0181       LDY     START,PCR   reset buffer pointer
01A8 39                 RTS

                01AA  DONE  EQU   *

                * dump buffer to disk

01AA BD 0C              BSR     CLOSBUF
01AC 86 04              LDA     #4          close the file
01AE A7 84              STA     ,X
01B0 BD D406            JSR     FMS         tell flex to do it

01B3 2  96              BNE     ERROR
01B5 1a F630            LBRA    AGAIN       and do it again

01B8 10AF 8D 0029 CLOSBUF STY   TEMP,PCR
01BD 30 8D 0029         LEAX    FCB,PCR     point to FCB
01C1 31 8D D165         LEAY    START,PCR   Get start of text buffer
01C5 A6 A0      BLOOP1  LDA     ,Y+         and get char
01C7 BD D406            JSR     FMS         give character to FLEX
01CA 1026 FF7D          LBNE    ERROR       exit if error
01CE 10AC 8D 0013       CMPY    TEMP,PCR    and continue till
0103 26 F0              BNE     BLOOP1      buffer empty
0105 39                 RTS

0106 65 6E 7a 63 MSG1   FCC     /enter filename:/
01E5 04                 FCB     4
01E6                TEMP    RMB     2
01E8                MEMMA1  RMB     2
01EA                FCB     RMB     320
                0328  START   EQU   *           start of memory buffer
                        END     MAIN

0 ERROR(S) DETECTED
```

913 TULBAGH CENTRE
HANS STRIJDOM AVENUE
CAPE TOWN
P.O. BOX 6829
8017 ROGGEBAAI
TEL. 21 6390

Dear Don

I think I've come across a potential problem with 6809 Flex which may be of interest to your readers.

The problem will only arise if

1. You are running a SWTP DAT, a DMA disk controller, and extended addressing.

2. your memory does not all reside in page 0 (i.e. the top 4 bits of the 20 bit address are not 0). e.g. say 32K in page 0 and 32K in page 1.

Under the above circumstances, if your FCB buffer is across a page boundary you will have a problem.

The reason is that although the processor board does its addressing through the DAT, the DMA controller does not. This means that if the high order 4 bits should change during the DMA process, they will not in fact change.

The easy solution is to make sure that your FCB buffer does not cross a page boundary.

Normally the problem would come up where 2 (or more) 32K memory boards are in a system and each has a full complement of chips.

By the way, the flex user guide specifically states that the FCB can reside at any address. But I think it was written prior to the 20 bit address availability.

Yours faithfully

ROB BERNSTEIN

Dear Don

Further to my letter of 14 February 1983, I've had confirmation from Mr Vanada at TSC that the problem exists.

The solution is not too difficult, it requires a patch to the disk drivers so that they do I/O thru a buffer area, and transfer the buffer from the program's disk I/O area.

Attached please find a copy of a little program which does the above.

Regards

ROB BERNSTEIN

RB/lvd

```
        NAM FIXIO
        ORG $0000

RTN     EQU *
RDCALL  PSHS X          SAVE X FOR LATER
        LEAX RTN,PCR    SETUP DMA ADDRESS AREA
        LEAX -256,X     GET DMA ADDRESS
RDJMP   JSR $FFFF       GO DO ACTUAL READ
        PULS X          GET ADDRESS OF CALLER'S MEMORY
        PSHS X,Y,U,D,CC SAVE 'EM ALL
        LDY #128        LENGTH OF MOVE
        LEAU RTN,PCR    SETUP DMA ADDRESS AREA
        LEAU -256,U     GET DMA ADDRESS
DMOV    LDD ,U++        GET 2 BYTES
        STD ,X++        PUT THEM IN CALLER'S SECTOR AREA
        LEAY -1,Y       DEC U
        BNE DMOV
        PULS X,Y,U,D,CC,PC

WRCALL  PSHS X,Y,U,D    SAVE THE REGS
        LEAU RTN,PCR    SETUP DMA ADDRESS AREA
        LEAU -256,U     GET DMA ADDRESS
        LDY #128        LENGTH OF A SECTOR, 2 AT A TIME
WMOV    LDD ,X++        WHERE IT COMES FROM
        STD ,U++        WHERE IT GOES TO
        LEAY -1,Y       DEC COUNTER
        BNE WMOV        ARE WE ALL DONE?
        PULS X,Y,D,U    RESTORE THE REGS
        PSHS X
        LEAX RTN,PCR    SETUP DMA ADDRESS AREA
        LEAX -256,X     GET DMA ADDRESS
WRJMP   JSR $FFFF       CALL ACTUAL WRITE
        PULS X,PC       RESTORE X & RETURN

RTNEND  EQU *
RTNLEN  EQU RTNEND-RTN

START   LDX $CC2B       MEMORY END
        LEAX -RTNLEN,X  ROUTINE SIZE
        LEAX -256,X     MAKE ROOM FOR DMA
        TFR X,D         PUT X INTO D
        CLRB            MAKE MODULO 256
        TFR D,X         PUT D INTO X
        LEAX -1,X       GET END OF FREE MEM ADDRESS
        STX $CC2B       SET NEW END OF MEMORY ADDRESS
        LEAX 1,X        SET X TO DMA AREA
        LEAX 256,X      GET ROUTINE START ADDRESS AREA
        PSHS X          SAVE X
        LDD $D601       GET READ VECTOR ADDRESS
        STD RDJMP+1     SET CALL READ ADDRESS
        LEAU RDCALL,X   ADDRESS WHERE RDCALL WILL RESIDE
        STU $D601       RESET VECTOR ADDRESS
        LDD $D604       GET WRITE VECTOR ADDRESS
        STD WRJMP+1     SET CALL WRITE ADDRESS
        LEAU WRCALL,X   ADDRESS WHERE WRCALL WILL RESIDE
        STU $D604       RESET VECTOR ADDRESS

        LDY #RTN        GET START OF ROUTINE
MOVIT   LDA ,Y+         GET A BYTE OF ROUTINE
        STA ,X+
        CMPY #RTNEND
        BLE MOVIT
        JMP $CD03       GO WARM START

        END START
```

David S Lapointe
1119 W. Summit
San Antonio, Texas 78201
(512) 732-6876

January 4,1983

Dear Don,

Well, I have been a subscriber to 68-Micro Journal for nearly two years and I haven't ever written you to tell you what a great magazine it is that you publish. It really is.

I have recently moved to San Antonio,Texas, and I have been hitting all of the bulletin boards trying to find other 68XX users with whom to exchange useful facts and fun. Do you know of any groups in the San Antonio area? Just out of curiosity I called SWTPc and they were nice enough to show me around the place, etc, but really didn't know of any way to put me in touch with people. They suggested that I write to your magazine. Perhaps a classified would do it?

Recently,I have moved to the San Antonio area and I would like to get in touch with people in the area who enjoy recreational as well as serious programming. D. Lapointe, 1119 W. Summit, San Antonio,TX 78201, (512) 732-6876 (voice).

47 Collingwood Road
WITHAM, Essex
CM8 2DZ
England
5th February 1983

Dear Don,

Here is the latest in my modifications to the TSC Editor. The version which I use is the cassette version, modified to accept FLEX commands as described long ago in your excellent journal. One of the problems associated with using the line-oriented editor, is remembering to insert carriage returns when typing in large amounts of text. This section of code modifies the Editor by inserting a carriage return in place of a space, once a pre-determined line length has been reached. In order to avoid over-running the input buffer, the minimum line length has been set to 110 characters, which allows the last word on the line to be 26 letters long, which I think is likely to be sufficient for all normal purposes. However, it can easily be set to some other value at assembly time.

If used with the "DC" command to use FLEX from within the Editor, then one only has to enter

@DC BET.AUTOEDIT

to enter the automatic mode. In this case, the original code (BSR BUFLIM) can be saved as UN-AUTO.BIN, so that the original form can be restored.

It should be noted that the routine cannot be made as a subroutine, as the return point is not to the code following the original JSR. Also the original BUFLIM subroutine must be preserved, as it is used elsewhere. Normal termination of a line by entering a carriage return, before minimum line length is reached is not affected, and all the usual editing facilities are retained. When using disk versions I think it will be necessary to patch, and use the entire EDIT.CMD, and of course there will be differences in locations, e.g. INBE63 may be at $0D45, and BUFFER at $00BD.

For anyone who writes a lot of text using this particular editor I think that this modification would be well worth implementing.

Yours sincerely,

Bill Hughes

```
                * AUTO-INSERT FOR TSC EDITOR *
                * WRITTEN BY W.A. HUGHES *
                        NAM     AUTOEDIT
00BB            BUFFER  EQU     $00BB
0B76            INBE63  EQU     $0B76   (JSR BUFLIM)
0B6B            INBE62  EQU     $0B6B   (JSR INCHAR)
0B7B            INBE67  EQU     $0B7B   (STAA 0,X)
1600            FREEPC  EQU     $1600   FIRST FREE SPACE
006E            MINLIN  EQU     110     (MINIMUM LINE LENGTH)
                *
0B76                    ORG     INBE63
0B76 7E 16 00           JMP     LIMIT
                *
1600                    ORG     FREEPC
1600 08         LIMIT   INX             BUMP POINTER
1601 8C 01 29           CPX     BUFFER+MINLIN       LINE TERMINATED
1604 2C 03              BGE     OVER                    AFTER MINIMUM
1606 7E 0B 6B   RETURN  JMP     INBE62                      LINE LENGTH
1609 81 20      OVER    CMPA    #2      CHANGE SPACE TO C/R
160B 26 F9              BNE     RETURN
160D 86 0D      CHANGE  LDAA    #0D     (C/R)
160F 09                 DEX             POINT TO CURRENT CHARACTER
1610 7E 0B 7B           JMP     INBE67
                        END
        NO ERROR(S) DETECTED


   SYMBOL TABLE:
BUFFER  00BB    CHANGE  160D    FREEPC  1600    INBE62  0B6B
INBE63  0B76    INBE67  0B7B    LIMIT   1600    MINLIN  006E
OVER    1609    RETURN  1606
```

```
$1060B767E1600E4
$1131A0008BC01202C037E0B6B0:2026F9B60D0999
```

intrel corp. [illegible]

COMPARING THE 6809 WITH THE Z80 AND THE 68000

Dear Dan,

The letter from Robert Peirce in the February issue (pg.39) leaves the reader with the very mistaken impression that Intrel-C on the 6809 finishes a poor third on the Eratosthenes sieve benchmark in comparison to Whitesmith's C on the Z80 and Unix C on the 68000.

The fact of the matter is that Mr. Peirce failed to note he was comparing performance figures obtained on a 1 Mhz 6809, a 4 Mhz Z80, and a 10 Mhz 68000. This is not a valid or fair comparison since, in terms of memory cycle times, a 2 Mhz 6809 is much more nearly equivalent to the 4 Mhz Z80 and 10 Mhz 68000 processors.

Using the current release of Intrel-C, which is a bit more efficient than the original version brought out a year ago, and running the sieve on a 2 Mhz 6809, the comperative performance is more accurately given by:

Unix C on the 10 Mhz 68000---------- 8.3 seconds
Intrel-C on the 2 Mhz 6809 ---------- 10.0 seconds
Whitesmith's C on the 4 Mhz Z80 ---------- 15.5 seconds

These more representative performance figures, of course, put an entirely different perspective on things than was suggested by the figures given in Mr. Peirce's letter. For one, in the 8-bit arena, Intrel-C on the6809 quite handily outperforms Whitesmith's C on the Z80. Secondly, Unix C on the 68000 is not as dramatically superior to Intrel-C on the 6809 as one might reasonably expect, given the fact that the 68000's 16-bit architecture gives it a decided advantage over the 8-bit 6809.

The 6809 is an excellent processor. Let's give it the credit it deserves!

Sincerely,

John Wisialowski

PTR-3 REVIEW

MDOS COMPATIBLE PARALLEL/SERIAL INTERFACE

My company recently purchased the Concurrent Technologies Corp. PTR-3 MDOS compatible Parallel/Serial Printer Interface Board. Perhaps a brief historic background of Concurrent Techologies would be helpful before we delve into the technical aspects of the board.

While a professor at Greenville Technical College, Bob Ehlers saw a need for a digital voltmeter. Working part time and consulting with local engineers, he built a prototype meter; however, it was never marketed because these same engineers were looking for someone to market a daisy wheel printer control board based on the Motorola 6802 MPU. Several other boards followed in quick succession and today Concurrent Technologies supports Bob and six other employees. What about the meter? Well, maybe someday...

This experience led Bob to address that nagging problem those of us that use the Motorola EXORciser® development system or systems designed with the Motorola Micro Module board face; and that is how to use an inexpensive, off the shelf, serial printer when Motorola software supports only a parallel printer? Until now, memory overlays and/or software patches have been the standard approach to obtaining a hard copy output. The PTR-3 board answers the need for a simple solution.

The board provides many pleasant surprises. It comes solder masked front and back, silk screened with component identification, and gold plated edge connectors. This is pretty much standard for a professionally produced board, but to find every IC residing in a socket is most unusual and quite welcome. Those of us who troubleshoot and repair boards find that it may end up costing 100 times (or more) the 24 cent cost of the defective IC that is soldered to the board. It is infinitely more economical to test and replace a socketed IC. The interface to the printer is through card edge connectors and ribbon cables. Again, standard; however you still have to find the right connectors and ribbon cable, right? rong! Packaged with the board are two cables. This means that all you have to do is configure the board and provide the desired printer. So much for the board cosmetics - lets get on to the layout.

There are three main sections on the board. These are the bus interface section, the Centronics type parallel interface section, and the RS-232-C/20 mA current loop serial section.

The bus interface section consists of address comparators, and data and control bus buffers. The board is shipped configured for hex address $EC10. This is the standard MDOS printer location. The user may easily change the address of the board by cutting copper runs on the foil side of the board and replacing them with header jumper blocks wired for the desired address.

The parallel connection is for a standard Centronics interface. To use this interface simply configure the jumper selectable options as described in the documentation and connect the interface cable supplied and you're ready to go.

The serial section of the board is interfaced through a ribbon cable (also supplied) terminated with a 25 pin female D-subminiature connector. This is the standard RS-232-C connection. There are eight standard baud rates provided, from 110 to 9600. At the higher baud rates, serial devices require handshake protocols to properly handle data which is coming in faster than the device can accept it. For this handshake function, two different protocols are provided by the interface. There is the hardware handshake which is the CTS line into the board. This line must be at an RS-232-C low level if the interface is to send characters to the printer. There is also a circuit to support the DC1/DC3 (XON/XOFF) protocol. If the printer being used does not support either of these protocols, the baud rate must be set low enough for the printer to completely handle each character before another character is sent down the line. The serial interface can also provide a 20 mA current loop port when proper connections are made with the on board jumpers. In this mode, handshaking is done with the DC1/DC3 protocol only.

Additional jumpers are provide to select the computer bus interface signals. J1 selects VMA, VUA, or VXA from the EXORciser bus. J2 selects IRQ, NMI, or the 6809 FIRQ to be generated. J3 is the memory page enable line for use in a non-standard system configuration.

LED's are provided to show the status of the CTS line or the state of XON/XOFF if that mode of handshaking is used. This LED is not

used in the parallel mode. The other two LED's show the state of the transmit and receive data lines. These LED's are not used in the parallel mode.

Several limitations become apparent in using the PTR-3 board. First, if you try to use it with a DEC LA 120, the CTS line signal from the DEC writer is inverted from the sense needed by the PTR-3 board. You must correct the line by using a unity gain op-amp circuit or other means to do the same thing. Bob tells me that this will be corrected on future versions of the board by providing an inverter on the board and a jumper to select the correct state for CTS.

The other problem involves an Epson printer. The symptom is that only every other character is printed. What is happening is that the Epson printer is designed as a single buffer receive line. Normally, most printers are double buffered. What all this means is that one character is being sent while the last transmission is in the receive buffer. To solve this problem you must cut the trace on the foil side of the board connected to pin 24 of the UART and connect it to pin 22.

In summary, the board is well designed and reliable. It is warranted for 6 months and is sold assembled and tested for $349 or as a bare board for $64.

Allen K. Thomas
Union Carbide Corporation
P.O Box 5928
Greenville, SC 29606

** QUALITY SOFTWARE NEEDED **
Standard S50 Bus and Color Computer

For the past few months we at the DATA-COMP Division of Computer Publishing, Inc. (CPI), the parent company of 68 MICRO JOURNAL, have debated expanding into the software distribution business. Many other magazines have been doing so for years. Presently there are many fine examples of software that has been developed by YOU our readers, that will never see the 'light of day' unless someone, with enough exposure and willingness to continually advertise, runs with the ball.

Software is the 'backbone' for the real utilization of any computer, ours are no exceptions! Realizing that there will be some conflicts, with other advertisers, this has been no simple decision. However, since day one the foremost concern of 68 MICRO JOURNAL has been it's readers! Therefore, DATA-COMP Division will accept, for appraisal, software that runs on 6809 systems, games, utility or applications programs.

In the past there has been too much software offered that was not quite ready, nearly, but not quite. We will strive to eliminate that element. But right up front we tell you only that we will do our very best, nothing more. Also we will strive to keep cost to a bare minimum, while securing for the author a fair return, in royalty payments, promptly paid.

Of course we will expect, no - demand, that the author keep the product free of errors (bugs), and maintain it on a prompt and business like basis. Also we shall require that authors be willing to furnish 'source' for those programs that justify, by price and utility, inclusion of same. The lack of source code, properly commented, is a continual complaint we hear. Not all programs will be sold with source, but where necessary, we will insist that it be included.

In some instances the program may be small or short and not justify itself as a 'single' sale product. In this event it will be combined with other like programs, and offered as a package. In that event the royalties will be split between the various authors.

If you have software that you feel will qualify under this program please contact the proper person as shown below.

Standard S50 Bus
Don Williams
Bob Nay

Color Computer
Tom Williams
Bob Nay

Remember, if your software has any problems or 'funnies' - GET IT STRAIGHT BEFORE YOU CONTACT US!!!!! Also get your source code in proper shape and well commented. There is too much 99% code already drifting around.

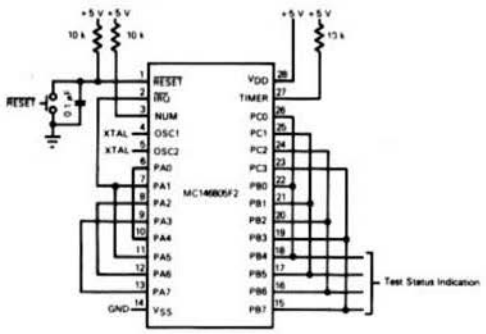
DATA-COMP, POB 796, Hixson, TN 37343 - (615) 842-4601
A Division of CPI

ERRATA
to
MC146805F2
ADVANCE INFORMATION DATA SHEET
(ADI-878)

In showing the external connections needed to perform the self-check program, the interrupt pin (pin 2) should be connected to PA1 (pin 7) and not PA0 (pin 6). This connection was erroneously shown in the self-check circuit diagram of the MC146805F2 Advance Information Data Sheet. The following diagram has been corrected and replaces Figure 10 (page 9) of the data sheet.

This change affects only those using edge- and level-triggered interrupt parts. The change will not affect the operation of the user-available peripherals as they require no external connections.

FIGURE 10 — SELF-CHECK PINOUT CONFIGURATION



MOTOROLA Semiconductor Products Inc.

Sirs, Do you know if anyone makes or is planning to make an 8" disk controller for the TRS-80C?

With a major investment in 8" DSDD QUME drives, it's a shame to have to make another investment in 5" drives, especially with reduced capacity.....(I have over 1 megabyte of system software on drive 0, and keep applications software on ONLY drive 1.) I would use 5" disk only for compatability access.

Thanks, s/John Nierste

P.S. Stylograph is GREAT!!! The best I've seen!!

John, reference your card of the 1st. I do not know of any 8" controller for the color computer, at the present time. I hear rumors of a controller, for the CoCo, that will handle both 5 and 8 inch disk, but nothing now.

It should not be too hard to interface the 8" disk to the present board, but, the ROM would need changing. However, with 80 track drives you can get right at 2800 FLEX sectors to a disk.

DMW - - -

CLASSIFIED ADVERTISING

Need good Engineer or Technician for 6809/68000 petrochemical projects in well established company, in Los Angeles. MOJAVE SCADA SYSTEMS 7100 Hayvenhurst Ave., Van Nuys, CA 91406 (213) 902-1611.

***

SWTPC: 1-MP-8M2 8K Static Memory Board $50; 1-MP-32S Universal Static Memory Board (never used & Unpopulated) $85.00; 1-MP-S Serial Board $25.00; 1-MP-P2 Parallel 2 Port board (never used) $85.00; 1-Percom CIS-30+ Tape interface $35 (never used); 1-Glmix Static Memory Board 32K supports extended addressing fully populated $225. Craig Cales, 805-962-9163 Days or 682-3597 Eves, 17 E. Carrillo Street. Suite #39, Santa Barbara, CA 93101.

HELP

Thomas Mason
2402 Audubon Road
Akron, OH 44320

HELP 68 Micro Journal

I am looking for a 6800 source listing for a terminal program that will let me transfer files from memory to my mod and from my modem to memory.
I have a Percom SBC/02 without mass storage that I use as an enhanced electric typewriter--not really word processing. So far, I just run my terminal straight into the modem for communications. I'd rather keep my micro on-line. Percom, as you know, has non-standard port assignments.

***

Bruce Cox
2287 Grange Hall Road
Dayton, Ohio 45431

Hi,

I need some HELP. Does anyone still run CP/68? I am trying to find someone to correspond with. I am running a 6800 with SSB and FLEX and I want to get up on CP/68. I am keying in all the code from the books, but if someone has already done this it will save me a lot of time.
Is anyone submitting 6800 articles anymore?

Editor's Note: Sorry Bruce but we get practically nothing for the 6800 these days. Have not had anything for the CP/68 system submitted in over 3 years. The 6809 and soon the 68000 are the popular CPUs now. We would still run some tape and 6800 for you folks but just don't get it. Sorry.

DMW

***

Finally the barrier has been removed from OS9™ to FLEX™ formatted disk!! Now you can **READ** and **WRITE** to a FLEX™ diskette, 5 or 8 inch, with **O-F**.

O-F is a new and *unique* program, written in BASIC09™ that performs the following functions, and comes complete with source.

1. **REFORMAT:** This module formats a disk that can be read by both OS9™ and FLEX™. Eight or five inch selectable.

2. **FLEX.BAS:** This program does the actual read or write function to the special O-F disk. Also it has the disk format and DIR (OS9) commands. All selectable from a user-friendly menu. All selections are interactive and complete including all necessary prompts to the operator.

3. **BFLEX.BAS:** This program allows binary programs to be exchanged, as FLEX.BAS above.

4. **DIR:** This module (menu selected) allows the disk directory to be printed to the screen, while in BASIC09.

FLEX users can read, write and use the special disk as any other FLEX disk, provided the FLEX directory is not allowed to continue beyond track zero (too many files).

$79.95

MasterCard VISA

DATA-COMP
P.O. Box 794 HIXSON, TN 37343
1-615-842-4601

Universal Data Research, Inc. Introduces

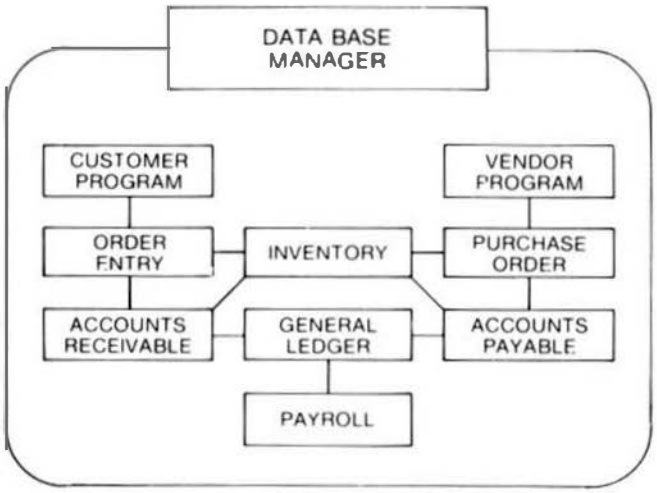
MODEMS

300 Baud Acoustic	$149.00
300 Baud Direct	$179.00
300 Baud Auto Answer	$219.00
1200 Baud Direct	$449.00
1200 Baud Auto Answer	$499.00
300/1200 Auto Answer	$549.00
300/1200 Auto Dial	$599.00
Modem 2 Port I/O Card	$119.00
with speed select	
Printer Special: Okidata 82A	$449

Dealer Inquiries Welcome ● Call or Write For Free Catalogue

New, Lower Prices

FLEX* and UniFLEX*
software for the 6809



Integrated Business Programs

	FLEX	UniFLEX
Accounts Payable	$295	$395
Accounts Receivable	$295	$395
General Ledger	$295	$395
Inventory 2	$295	$395
Payroll	$295	$395
Data Base Manager	$350	$450
Word Processing		
Software	$295	$395
WP Menu		$150
P Control		$150

UNIVERSAL DATA RESEARCH INC.

*FLEX & UniFLEX are Trademarks of Technical Systems

2457 Wehrle Drive, D-68, Buffalo, NY 14221
PHONE (716) 631-3011

Dealer Inquires Welcome

master charge VISA

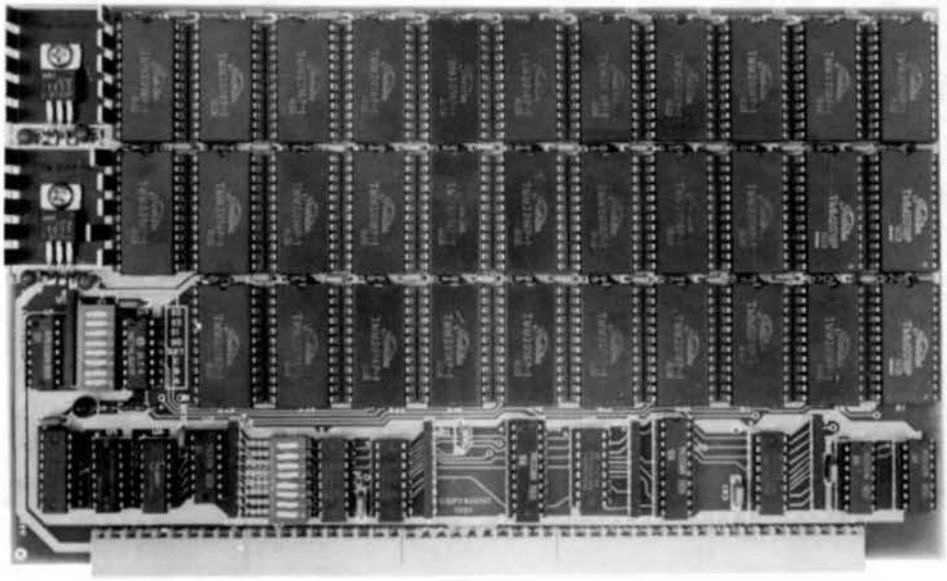
64K SS-50 STATIC RAM

NEW!

$199.00 (48K KIT)

NEW!

LOW POWER!



RAM OR EPROM!

BLANK PC BOARD WITH DOCUMENTATION $52

SUPPORT ICs · CAPS -	$18.00
FULL SOCKET SET -	$15.00

ASSEMBLED AND TESTED **ADD $40**

FEATURES:

56K	$249
64K	$299

- ★ Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
- ★ Fully supports Extended Addressing.
- ★ 64K draws only approximately 500 MA.
- ★ 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
- ★ Board is configured as 3-16K blocks and 8-2K blocks (within any 64K block) for maximum flexibility.
- ★ 2716 EPROMs may be installed anywhere on Board.
- ★ Top 16K may be disabled in 2K blocks to avoid any I/O conflicts.
- ★ One Board supports both RAM and EPROM.
- ★ RAM supports 2MHZ operation at no extra charge!
- ★ Board may be partially populated in 16K increments.

16K STATIC RAMS?

The new 2K x 8, 24 PIN, static RAMs are the next generation of high density, high speed, low power, RAMs. Pioneered by such companies as HITACHI and TOSHIBA, and soon to be second sourced by most major U.S. manufacturers, these ultra low power parts, feature 2716 compatible pin out. Thus fully interchangeable ROM/RAM boards are at last a reality, and you get BLINDING speed and LOW power thrown in for virtually nothing.

Digital Research Computers
(OF TEXAS)
P.O. BOX 401565 • GARLAND, TEXAS 75040 • (214) 271-3538

TERMS: Add $2.00 postage. We pay balance. Order under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50, add 85¢ for insurance.

ARCADE 50

POWERFUL COLOR GRAPHICS
Uses the new TMS9918A Video Display Processor. High resolution 256 x 192 pixel display with 15 colors 16k Bytes of onboard RAM does not reduce user memory. 32 graphic images can be individually moved with simple X-Y commands for smooth animation.
External Video input allows subtitling
NTSC composite video output
SOUND EFFECTS AND MUSIC
ThreeAY3-8910 Programmable Sound Generators
Nine simultaneous voices
Three independent noise sources
Onboard stereo amplifier drives two 8 ohm speakers
ADDITIONAL I/O CAPABILITIES
Eight analog inputs with 8 bit resolution
Supports four joysticks with pushbutton switches
Eight bit parallel I/O port
Entire unit maps into 256 bytes of memory
DOCUMENTATION AND SOFTWARE
Programming manuals for Video and Sound Processors
Subroutine library and Super Demo Maze Game
Example programs in BASIC, FBASIC and ASSEMBLY
User library and sales support

ARCADE 50 assembled and tested	$325.00
Video and Audio connector set	15.00
4 Joystick connector set	15.00
2 Radio Shack Joysticks	24.00
UHF channel 33 modulator	32.00
Gold Molex connectors	12.00
A/BASIC for 6800	110.00
FBASIC for 6809	110.00
FBASIC (with ARCADE 50)	75.00
FBASIC (manual only)	10.00
ARCADE 80 (TRS Model I)	395.00
ARCADE 100 (S-100 BUSS)	375.00
ARCADE 50 RGB	375.00
LABVIDEO (Motorola EXORbus)	375.00
LABVIDEO RGB	375.00
NEW MV09 6809 Processor Board	225.00
*Comes assembled with PIA and ACIA	
*12 Sockets for 2716, 2732 or RAM	
*Supports DMA disk I/O	
*Ideal for 6809 upgrade or process control	
AMDEK COLOR I Monitor	425.00
AMDEK COLOR II Monitor	799.00
AMDEK COLOR III Monitor	499.00
256K Dynamic Memory Board (assembled)	795.00
256K Dynamic Memory Board (assembled w/64K)	395.00
64K Dynamic Memory Board (assembled)	295.00

Specify 5" or 8" soft sector disk for TSC's FLEX or MICROWARE'S OS/9 system
TERMS: CASH, VISA, MC, C.O.D.

FBASIC

TERMINUS DESIGN INC., in conjunction with Microware Systems Corporation, is proud to announce FBASIC-an enhancement of Microware's 6800 A/BASIC. Their fast compiled BASIC has been adapted for 6809 users with added video and sound features for ARCADE 50 users. FBASIC is a true compiler that produces optimized machine language modules which are ROMable and require no Run-Time package. FBASIC requires less memory overhead and runs hundreds of times faster than BASIC interpreters. It supports standard BASIC instruction including String functions, Disk I/O and fast integer arithmetic with multiple-precision capability. Graphics verbs and functions fully support the Arcade 50. Arcade statements include:

INIT	MODE	BLANK	BACKDROP
SIZE	MAG	VREG	DELAY
MOVE	DRAW	FCOLOR	JSWITCH
REMOVE	RDRAW	BCOLOR	SWITCH
PSG	TONE	ENVL	VOLUME
ADC	SPRITE	SPNAME	ENDEF
SPCOLOR	RSPRITE	SPDEF	PATDEF
VPEEK	VPOKE	VPRINT	

TERMINUS DESIGN INC
16 SCARBROUGH ROAD
ELLENWOOD, GA 30049
(404) 474-6866

68 MICRO JOURNAL PROGRAMS on DISK

Disk #1: FILESORT, MINICAT, MINICOPY, MINIFMS, **LIFETIME, **POETRY, **FOODLIST, **DIET.
Disk #2: DISKEDIT w/ Inst. & fixes, PRIME, *PRMOD, **SNOOPY, **FOOTBALL, **HEXPAWN, **LIFETIME.
Disk #3: CBUG09, SEC1, SEC2, FIND, TABLE2, INTEXT, DISK-EXP, *DISKSAVE.
Disk #4: MAILING PROGRAM, *FINDDAT, *CHANGE, *TESTDISK.
Disk #5: *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk #6: **PURCHASE ORDER, INDEX (Disk file indx).
Disk #7: Linking Loader & RLOAD, Harkness
Disk #8: CRTSET, Lanpher (May '82)
Disk #9: DATECOPY, DISKFIX9 (Aug '82)

NOTE: All are as published or received by 68 Micro Journal, some have fixes and patches.

This is a reader service only! No Warranty is offered or implied, they are as received and are for reader convenience ONLY. Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

PRICE: 8" Disk $19.95 - 5" Disk $17.95

68 MICRO JOURNAL
POB 794
Hixson, TN 37343
615-842-4600

* indicates 6800, ** indicates BASIC SWTPC or TSC - 6809 no indicator.

MASTER CARD - VISA accepted - Foreign add sufficient postage surface or air!!

WINCHESTER FOR MOTOROLA EXORCISOR/MDOS

☐ 10 MB Winchester hard disk runs MDOS on Motorola Exorcisor System ☐ No modification to MDOS required ☐ MDOS based software stays alive ☐ All user software operates without modification ☐ Optional SA-801R flexible diskette drive system ☐ Optional 10 MB removable cartridge.

CSA *For information call (619) 566-3911*
Computer System Associates
7562 Trade Street, San Diego, CA 92121

TEN MOST-ASKED QUESTIONS

ABOUT DYNACALC™

THE ELECTRONIC SPREAD-SHEET FOR 6809 COMPUTERS

1. **What is an electronic spread-sheet, anyway?**
   Business people use spread-sheets to organize columns and rows of figures. DYNACALC simulates the operation of a spread-sheet without the mess of paper and pencil. Of course, corrections and changes are a snap. Changing any entered value causes the whole spread-sheet to be re-calculated based on the new constants. This means that you can play, 'what if?' to your heart's content.

2. **Is DYNACALC just for accountants, then?**
   Not at all. DYNACALC can be used for just about any type of job. Not only numbers, but alphanumeric messages can be handled. Engineers and other technical users will love DYNACALC's sixteen-digit math and built-in scientific functions. There's even a built-in sort command, so you could use DYNACALC to manage small data bases - up to 256 records.

3. **What will DYNACALC do for ME?**
   That's a good question. Basically the answer is that DYNACALC will let your computer do just about anything you can imagine. Ask your friends who have VisiCalc, or a similar program, just how useful an electronic spread-sheet program can be for all types of household, business, engineering, and scientific applications.

4. **Do I have to learn computer programming?**
   NO! DYNACALC is designed to be used by non-programmers, but even a Ph.D. in Computer Science can understand it. Built-in HELP messages are provided for quick reference to operating instructions.

5. **Do I have to modify my system to use DYNACALC?**
   Nope. DYNACALC uses any standard 6809 configuration, so you don't have to spend money on another CPU board or waste time learning another operating system.

6. **Will DYNACALC read my existing data files?**
   You bet! DYNACALC has a beautifully simple method of reading and writing data files, so you can communicate both ways with other programs on your system, such as the Text Editor, Text Processor, Sort/Merge, RMS data base system, or other programs written in BASIC, C, PASCAL, FORTRAN, and so on.

7. **How fast is DYNACALC?**
   Very. Except for a few seldom-used commands, DYNACALC is memory-resident, so there is little disk I/O to slow things down. The whole data array (worksheet) is in memory, so access to any point is instantaneous. DYNACALC is 100% 6809 machine code for blistering speed.

8. **Is there a version of DYNACALC for MY system?**
   Probably. You need a 6809 computer (32k minimum) with FLEX or UniFLEX operating system. A version for OS-9 is also in the works. You also need a decent CRT terminal, one with at least 80 characters per line, and direct cursor addressing. If your terminal isn't smart enough for DYNACALC, you probably need a new one anyway. The UniFLEX version of DYNACALC also allows you to mix different brands of terminal on the same system. There's also a special version of DYNACALC for Color Computers equipped with FLEX and Data-Comp's F-MATE. A version for Frank Hogg's Color Computer FLEX is also being done.

9. **How much does DYNACALC cost?**
   The FLEX versions are just $200 per copy; UniFLEX version $395. Foreign orders add $10 per copy for postage. We encourage dealers to handle DYNACALC, since it's a product that sells instantly upon demonstration. Call or write on your company letterhead for more information.

10. **Where do I order DYNACALC?**
    See your local DYNACALC dealer, or order directly from CSC at the address below. We accept telephone orders from 10 a.m. to 6 p.m., Monday through Friday. Call us at 314-576-5020. Your VISA or MasterCard is welcome. Please specify diskette size for FLEX versions. Software serial number is required for the UniFLEX version of DYNACALC.

ORDER YOUR DYNACALC™ TODAY

Foreign Dealers:

Australia & Southeast Asia: order from Paris Radio Electronics, 7A Burton St., Darlinghurst, NSW 2010 Sydney. Telephone: 02-357-5111.

United Kingdom: order from Compusense, Ltd., PO Box 169, London N13 4HT. Telephone: 01-882-0681.

Scandinavia: order from Swedish Electronics hk AB, Murargatan 23-25. Uppsala S-754 37 Sweden. Telephone: 18-25-30-00.

Computer Systems Center
13461 Oliva Blvd.
Chesterfield, MO 63017
(314) 576-5020

OS-9 Version NOW Available

$250.00

UniFLEX software prices include maintenance for the first year.

DYNACALC, DYNAMITE, and DYNAMITE + are trademarks of Computer Systems Center.

F-MATE is a trademark of Data-Comp.
VisiCalc is a trademark of VisiCorp.
OS-9 is a trademark of Microware and Motorola.
FLEX and UniFLEX are trademarks of TSC.

ALSO FROM CSC

DYNAMITE +
"THE CODE BUSTER"

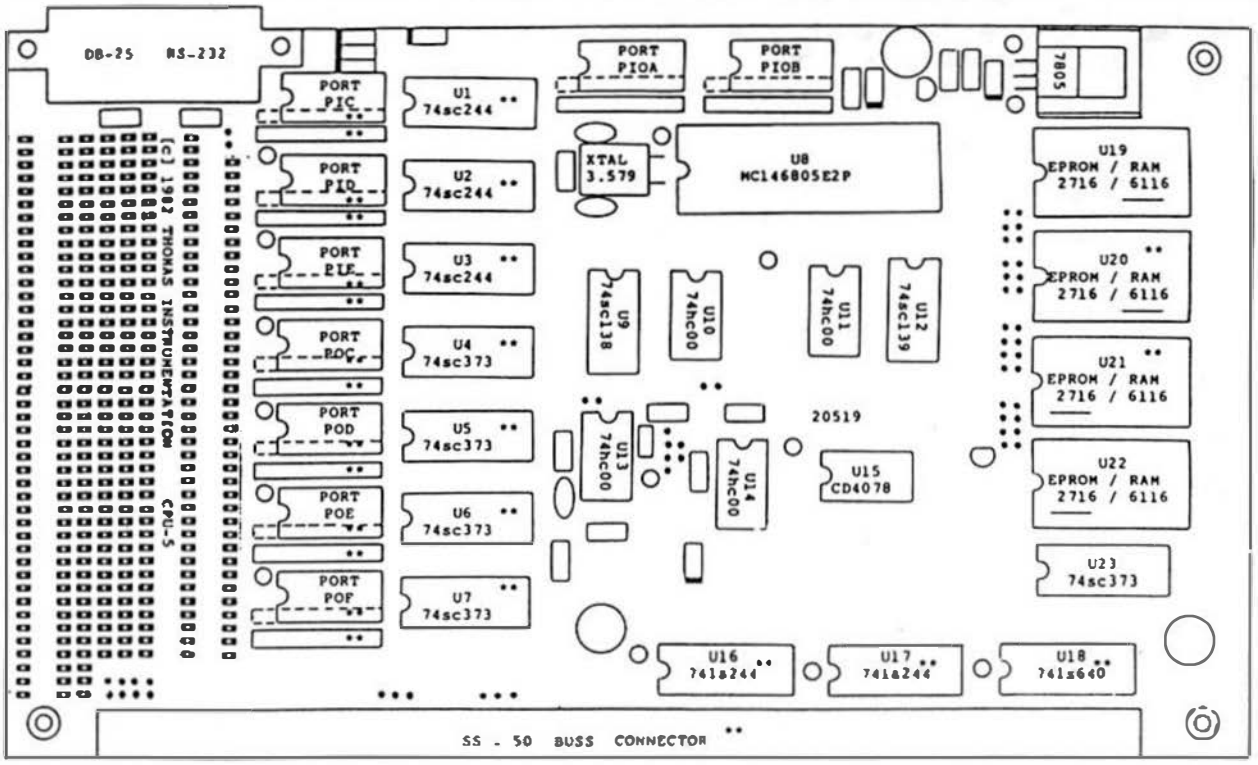
now available for UniFLEX OS-9 version soon

DYNAMITE + is a new version of DYNAMITE, our popular 6809/6800 disassembler package for 6809 FLEX. Present users of DYNAMITE can upgrade to DYNAMITE + by sending us the original DYNAMITE diskette and $40 (plus $5 for foreign postage). DYNAMITE + does everything DYNAMITE does, and more! A cross-reference generator has been added, label files are now maintained only in text form (LABEL EQU $xxxx), and boundary file specifications have been tremendously simplified, which makes it easier to disassemble large programs containing lots of big tables.

The UniFLEX version of DYNAMITE + does everything the FLEX version does, and also automatically handles system calls and 'info' areas.

DYNAMITE + is available for $100 per copy on FLEX (specify diskette size), and $300 on UniFLEX. Foreign orders add $5 per copy for postage.

*** NEW 6805 ALL CMOS CARD ***



CPU-5-B	Blank card with documentation	$49.00
CPU-5-P	Partial card (parts marked ** not incl.)	$229.00 } Includes EPROM Monitor
CPU-5-F	Full card with all parts shown (fully assembled)	$349.00 } and 115 vac Power Pak.
CROSS-ASSEMBLERS	for FLEX ((c) TSC) 6809 system	$150.00

SOFTWARE

VDISK Treat extended memory as a super fast disk drive

6809 source & object	$149.00
6809 object	$ 99.00

OUTSIDE MODEM PROGRAM incl. source

UniFLEX version	$100.00
FLEX version 6800 & 6809	$ 50.00
HAYES SMART MODEM	$249.00

CROSS ASSEMBLERS for 6800,6801,6805 runs on 6809 FLEX $150.00

TV-EDIT Screen oriented Editor for FLEX 6809 $ 95.00
Please specify 5" or 8" Disk when placing order.

CHIPS

Please note when ordering IC's that a $100.00 Minimum is in effect Orders less than the minimum will be charged an additional $10.00 for handling

6116P-3	2MHZ	$13.50
2016P-2	2MHZ	$10.50
2716	1MHZ	$ 6.50
2716-1	2MHZ	$16.50
146805E2P		$25.00
68B02		$10.00
68B09		$25.00
68B21		$ 6.50
68B40		$12.00
68B50		$ 6.75
74LS640		$ 3.25

HARDWARE

NEW MEMORY PRICES TO INTRODUCE VDISK

S-R/R without memory chips	$120.00
with 8K NMOS@2MHZ	$169.00
with 16K NMOS@2MHZ	$199.00
with 32K NMOS@2MHZ	$299.00
with 48K NMOS@2MHZ	$399.00
Bare Card	$ 49.00

Extender Cards assemb. with logic aid

SS-50/50C	$ 35.00
SS-30	$ 25.00
SP-1 Prototype Board A/T	$195.00
SP-1 Bare Card	$ 49.00
SS-50 Wire Wrap Board B/C	$ 39.00
SS-30 Wire Wrap Board B/C	$ 20.00
6802 Super CPU A/T	$235.00
6802 Super CPU B/C	$ 59.00
Video RAM, B/C $49.00, A/T	$195.00
Parallel I/O, B/C $49.00, A/T	$139.00
SS-50 Backplanes 4,6,8,12,&16 position @ $5.00 per slot w/o connectors	
SS-30 8 pos. BP w/o connect.	$ 39.00
Transition Card $49.00, A/T	$ 95.00
Molex Gold Male $1.60, Female	$ 1.60
Molex Tin Male $ .40, Female	$ .50

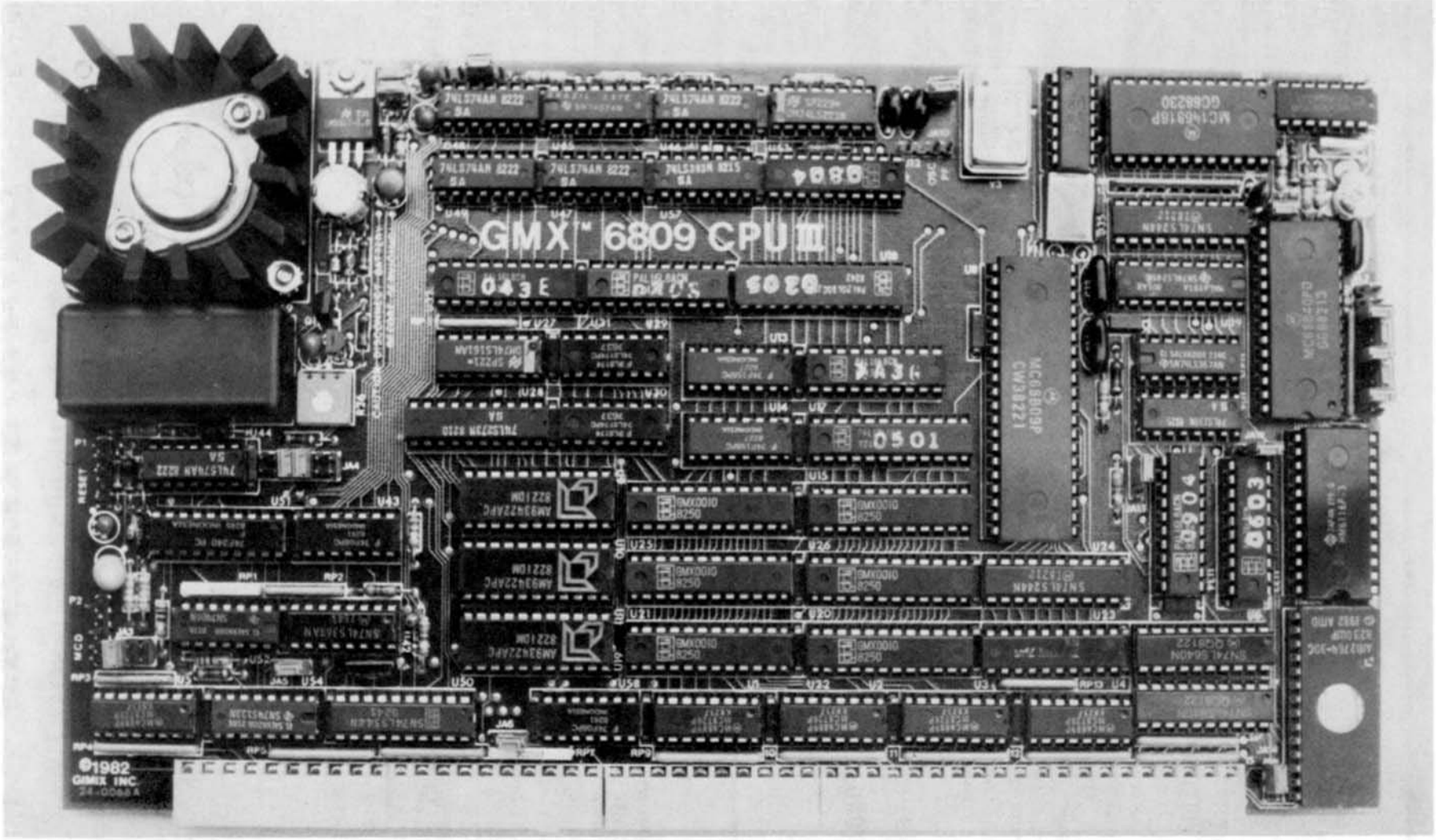
THOMAS INSTRUMENTATION

168 EIGHTH STREET — AVALON, N.J. 08202 (609) 967-4280
NJ RES. INCLUDE 5% SALES TAX
Allow 3 Weeks for Checks to Clear
CONT. USA INCLUDE $3.00 SHIPPING. CANADA $6.00, FOREIGN $12.00
MASTERCARD, VISA, and C.O.D. ACCEPTED
Send $20.00 to receive full documentation, schematics, & source listings for all boards currently in production.

'68' MICRO JOURNAL ADVERTISERS INDEX

This Index is provided as a reader service. The publisher does not assume any liability for omissions or errors.

The GIMIX CPU III Card



Watch this space for Systems featuring this CPU and GIMIX Intelligent I/O Cards

The Original FLEX for Color Computers

- Upgrade to 64K
- RS to FLEX, FLEX to RS file transfer ability
- Create your own character set
- Automatic recognition of single or double density and single or doubled sided
- All features available for either single or multiple drive systems
- Settable Disk Drive Seek Rates
- Faster High Resolution Video Display with 5 different formats
- Save RS Basic from RAM to Disk
- Move RS Basic to RAM
- Load and save function on FLEX disk
- 13 Support Commands 8 with Source Text

Languages Available

Pascal, Fortran, RS Basic, RS Assembler, TSC Basic, TSC Assembler, Relocating Assembler, Macro Assembler, Mumps

If you are tired of playing games on your TRS-80C™ Color Computer, or find that you are handicapped by the limitations of the RS BASIC in trying to write a Program that will allow you to actually USE the Color Computer as a COMPUTER, YOU ARE READY TO MOVE UP TO THE FLEX9™ Operating System. If you want to have REAL PROGRAMMING POWER, using an Extremely Powerful Business BASIC, PASCAL's, C Co pilers, a full-blown Macro Assembler with a Library capability so you are not continuously "reinventing the wheel", YOU ARE REA Y TO MOVE UP TO THE FLEX9™ Operating System. If you would like to see if YOU REALLY COULD USE A COMPUTER IN YOUR BUSINESS, or begin to make your Computer start PAYING IT'S OWN WAY by doing some Computer Work for the millions of small businesses around you, such as Wordprocessing, Payroll, Accounting, Inventory, etc., then YOU ARE READY TO MOVE UP TO THE FLEX9™ Operating System. How?? DATA-COMP has the way!

DATA-COMP's FLEX9™ Conversion for the TRS-80C™ Color Computer was designed for the SERIOUS COMPUTER **USER**; with features like greatly increased Display Screens, WITH Lower Case Letters, so you can put a FULL Menu on ONE Screen, or see SEVERAL Paragraphs at the same time; with features like providing a FULL Keyboard so you have FULL Control of your Computer AND it's Programs NATURALLY, without needing a chart to see what Key Combination will give you what function; with USER ORIENTED functions to make using the Operating System natural, like having the Computer AUTOMATICALLY determine what type of Disk is being used in what type of Disk Drive and working accordingly, rather that you have to specify each and every thing for it, or like having the Computer work with the Printer you have been using all along without you having to tell the new Operating System what is there; etc., etc., etc.

DATA-COMP has everything you need to make your TRS-80C™ Color Computer WORK for YOU; from Parts and Pieces to Full, Ready To Use SYSTEMS. DATA-COMP designs, sells, services, and SUPPORTS Computer SYSTEMS, not just Software. CALL DATA-COMP TO AY to make your Computer WORK FOR YOU!

System Requirements

FLEX9 Special General Version x Editor & Assembler (which normally sell for $50.00 ea)	$150.00
F-MATE(RS) FLEX9 Conversion Rout. for the RS Disk Controller	
when purchased with Special General FLEX9 Sys.	$89.95
when purchased without the General FLEX9 Sys.	$79.95
Set of Eight 64K RAM Chips w Mod. Instructions	$69.96
Color Computer with 64K RAM and EXT. BASIC	$499.95
Color Computer with 16K RAM	$289.95
Color Comput r with 16K RAM and EXT. BASIC	$389.95

SPECIAL SYSTEM PACKAGES

64K Radio Shack COLOR COMPUTER, Radio Shack COLOR DISK CONTROLLER, a Disk Drive System, Special General Version of FLEX9™, F-MATE(RS)™ and a Box of 10 Double Density Diskettes; a COMPLETE, ready to run SYSTEM on your Color TV Set. $1249.95

DISK DRIVE PACKAGES, etc.

These Packages include the Radio Shack Disk Controller, Disk Drives with Power Supply and Cabinet, and Disk Drive Cable

PAK #1 — 1 Single Sided, Double Density Sys.	$499.95
PAK #2 — 2 Single Sided, Double Density Sys.	$769.95
PAK #3 — 1 Double Sided, Double Density Sys.	$599.95
PAK #4 — 2 Double Sided, Double Density Sys.	$949.95
PAK #5 — 2 Qume Thinline Double Sided Double Density Sys.	$764.95

PARTS AND PIECES

Radio Shack Disk controller	$179.95
1 Tandon Single Sided, Double Density Disk Drive	$249.95
1 Tandon Double Sided, Double Density Disk Drive	$349.95
1 Qume Thinline Double Sided, Double Density	$279.95
Single Drive Cabinet with Power Supply	$89.95
Double Drive Cabinet with Power Supply	$109.95
Single Drive Disk Cable for RS Controller	$24.95
Double Drive Disk Cable for RS Controller	$34.95
Micro Tech. Produ., Inc. **LOWER CASE ROM** Adapter	$74.95
Radio Shack BASIC Version 1.1 ROM	$34.95
Radio Shack Extended Basic ROM	$89.95

TERM — AN External Terminal Driver is FREE with purchase of F-Mate

SOFTWARE

CHESS 6809

Requires FLEX™ and one of the following CRT terminals

Now Runs On Any Type Terminal

Features:

- Two display boards.
- Four levels of play.
- Point scoring system.
- Play white or black.
- Change or set-up boards piece positions.
- Forfeit move.
- Swap sides.
- Make move and swap sides.
- Change skill level.
- Stop and restart game.
- Solve 'Mate in 1-2-3-4' moves.

$79.95 Specify 5" or 8" disk

This is one of the strongest CHESS programs running on any microcomputer, estimated USCF Rating 1600 +.

Note: Personal checks allow 3-4 week delivery.

DIET-TRAC Forecaster

A Diet Planning and Analysis Program

DIET-TRAC Forecaster is a program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C.P.F. %) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual.

Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individuals are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. When a weight goal is given (either gain or loss), and a calorie plan is agreed upon between the computer and the individual, the number of days to reach the weight goal is projected. The starting and ending rate of weight loss is calculated, and a daily calender with each day's predicted weight for a 30-day period is printed.

FLEX VERSION	**$59.95**
UniFLEX VERSION	**$89.95**

PRINTERS

The Epson MX-80
$495.00

The Epson MX-100
$725.00

MX-70 $355.00 MX-80 FT $575.00

☆

Color Computer External Terminal Program

$19.95

☆

Color Computer Stylograph Running Under F-Mate

$195.00

MasterCard VISA

DATA-COMP SOUTH EAST MEDIA

P.O. Box 794 HIXSON, TN 37343

1-615-842-4601

Verbatim Diskettes.

5" Soft Sector Disks

Single Side Single Density	$2.75 ea.
Single Side Double Density	$2.75 ea.
Double Side Double Density	$4.92 ea.
Plastic Storage Box	$7.00 ea.

8" Soft Sector Disks

Single Side Single Density	$3.75 ea.
Single Side Double Density	$4.10 ea.
Double Side Double Density	$4.75 ea.
Plastic Library Box	$5.00 ea.

Foreign Orders Add 10% Surface—20% Air Mail

WINCHESTER BACKUP UTILITIES

The following utilities allow the backup of any size disk System to any size diskettes.

By simply inserting diskettes as requested by COPYMULT, a large disk system (Winchester, etc.) may be downloaded to your present floppy disk system, any size. No need to fiddle with directory deletions or any of the other tedious operations that must be done using a normal copy routine.

COPYMULT-CMD understands normal "copy" syntax and always keeps up with files already copied by maintaining directories for both host and receiving disk system, thus eliminating hours of tedious keyboard entries and other time consuming cleanup chores.

BACKUP-CMD is a special program that downloads "random" type files, any size.

RESTORE-CMD a special program to restructure copied "random" files for copying, or recopying back to the host system.

FREELINK-CMD a "bonus" utility that "relinks" the free chain of a floppy or hard disk thereby eliminating fragmentation.

** Completely documented source files included.

** ALL 3 Programs $99.50 on 8" diskette

Data-Comp—South East Media & 68 Micro Journal Are Divisions of CPI

68 Micro Journal
5900 Cassandra Smith
Hixson, TN 37343

Second Class Postage Paid
At Chattanooga, TN
ISSN 0194-5025